Thacker Network Technologies Inc.

MobileApp Designer Guide

SatelliteForms™
MobileApp Designer

For Palm OS® & Pocket PC Handhelds

Legal Notice

Thacker Network Technologies Inc., 5338 - 51 Ave., Lacombe, Alberta T4L 1N5 Canada.

http://www.SatelliteForms.net

Document revision history:

2005-04-28 -Updated references from Intellisync to Thacker Network Technologies Inc., version V6.0.1 release.

2005-10-26--Updated for SatForms 6.1.0 release.

2006-02-16--Updated for SatForms 6.1.1 release.

2006-10-10--Updated for SatForms 7.0 release.

2007-11-19--Updated for SatForms 7.1 release.

2008-05-23--Updated for SatForms 7.2 release.

2010-07-12--Updated for SatForms 8 release.

Contents

Preface

Chapter 1
What's New

Chapter 2
Satellite Forms Overview

Chapter 3
Installing Satellite Forms

Chapter 4
Quick Tour

Chapter 5
MobileApp Designer Reference

Chapter 6

Creating your Application

Chapter 7

Using Actions, Filters, Extensions, and Color

Chapter 8
Integrating with your Database

Chapter 9
Using Satellite Forms on Handheld Devices

Chapter 10
Deploying your Application

Chapter 11

Satellite Forms Scripting Language Reference

Chapter 12
Satellite Forms API Reference

Chapter 13
Sample Application: Work Order

Appendix A
Tips and Tricks

Examples

Figures

Tables

Preface

This preface provides information about:

- Who should read this guide
- Contents of this guide
- Other Satellite Forms® MobileApp Designer documentation
- Technical Support
- Documentation conventions used in this guide

Who should read this guide

Developers responsible for creating, maintaining, and installing custom applications created with Satellite Forms should read this guide.

What this guide contains

This guide contains the following chapters:

- Preface (this chapter)

  Provides and overview of the Satellite Forms Development Guide, documentation conventions, other useful documents, and technical support contact information.

- Chapter 1, What’s New

  Provides an overview new features in this release.

- Chapter 2, Satellite Forms Overview

  Describes the fundamental concepts of Satellite Forms. We strongly recommend that you read this chapter and the Quick Tour chapter before getting into the more detailed information on how to use MobileApp Designer. Understanding how Satellite Forms organizes and accesses information is essential for creating applications with MobileApp Designer.

- Chapter 3, Installing Satellite Forms

  Presents system requirements for using the development environment and instructions for installing and testing Satellite Forms.

- Chapter 4, Quick Tour

Presents a brief tutorial for using MobileApp Designer to design and build your first Satellite Forms application.

- Chapter 5, MobileApp Designer Reference

  Provides a complete reference to all MobileApp Designer visual tools.

- Chapter 6, Creating your Application

  Provides steps required for developing your Satellite Forms application.

- Chapter 7, Using Actions, Filters, Extensions, and Color

  Presents information regarding the use of actions, filters, and SFX extensions to enhance the power and flexibility of your Satellite Forms applications.

- Chapter 8, Integrating with your Database

  Presents information on how to integrate Satellite Forms with your database application.

- Chapter 9, Using Satellite Forms on Handheld Devices

  Explains how to use the Satellite Forms engine on handheld devices.

- Chapter 10, Deploying your Application

  Provides the lists and steps required for deploying your Satellite Forms application.

- Chapter 11, Satellite Forms Scripting Language Reference

  Provides a complete reference to the Satellite Forms scripting language, a Visual Basic-like language you can use to enhance the power and flexibility of your applications.

- Chapter 12, Satellite Forms API Reference

  Provides a complete reference to the Satellite Forms API. You can use the API functions described in this chapter to write your own SFX plug-in extensions and SFX Custom controls.

- Chapter 13, Sample Application: Work Order

  Presents a complete description of a real-world sample application.

- Appendix A, Tips and Tricks

  Offers Satellite Forms programming tips and tricks to help you in develop effective, efficient applications.

- Glossary

  A complete glossary of specialized terms employed in this guide.

Satellite Forms documentation

For additional information on working with MobileApp Designer to create custom applications, refer to the online help and to the guides listed below. Satellite Forms documentation is available on the Satellite Forms Installation CD and in the Docs directory in the MobileApp Designer installation directory.

- ***Satellite Forms MobileApp Designer Development Guide:*** (This document) Satellite Forms MobileApp Designer system requirements, upgrade, and installation information. Conceptual, procedural, and troubleshooting information on creating, deploying and maintaining custom applications.

  It is available in both searchable help (**SatFormsHelp.chm**) and Adobe PDF format (**SF_MobileApp_Guide.pdf**).

- **Readme:** Important Satellite Forms issues that are not included in other product documentation, in ASCII text format.

  \Satellite Forms 8\Doc\SatFormsReadMe.txt

- ***Satellite Forms KnowledgeBase:*** The KnowledgeBase (KB) contains technical articles including several How To guides, as well as known problems in the current release. It is a searchable help file named **SatFormsKB.chm**. Updated versions of the KnowledgeBase are searchable and downloadable online at our website: http://www.satelliteforms.net/knowledgebase.htm

- ***Satellite Forms Solutions Guide:*** The Solutions Guide is a searchable database of collection of solutions for SatForms developers, organized into various categories. It is named **SatFormsSolutionsGuide.chm.** The intent is to provide a comprehensive directory of solutions and solution providers that Satellite Forms developers can access to enhance their handheld applications. Updated versions of the Solutions Guide are searchable and downloadable online at our website: http://www.satelliteforms.net/solutions.htm

Document conventions

This document uses the following conventions:

- Important information and application names appear in **bold** text, as shown in the following example: **Guidgen.exe**

- Interface items you must select or click are in **boldface** font.

- Information or code you must type, as well as function, event, and defined names are in `monospace` font.

- Menu selections are indicated as follows: **File > Import > MPX**.

- Variables are in *italic* font and may be enclosed in greater than and less than symbols, for example, <*server name*>.

  Substitute the correct information for the specified variable before executing the command.

- If you are viewing this document online, hyperlinks appear in blue text.

Technical Support

There are several sources of information that can help you get the answers you need. Please complete the following steps before contacting technical support:

1 Review the Readme file. The Readme file is updated with the latest information for each new version of the program. You can find the Readme file on the product CD and in the installation directory.

2 Review this guide. It contains information on system requirements, upgrading, and configuring Satellite Forms MobileApp Designer.

3 If the product installed successfully, but there are other issues:

   - Use the online help system installed with MobileApp Designer.

4 Visit the Satellite Forms web site at http://www.satelliteforms.net/.

5 Join the SatFormsDev Discussion Forum at http://www.satelliteforms.net/forum.htm for an online community of Satellite Forms developers.

Contacting Technical Support

Satellite Forms is registered automatically when you purchase the software. If you have reviewed the product documentation and are still experiencing difficulties with Satellite Forms, go to the Support Request on the website at http://www.satelliteforms.net/supportform.htm. On the **Support Files** page, visit the KnowledgeBase or other links.

When you contact Technical Support, please have the following information available:

- The Satellite Forms MobileApp Designer license key. This license key was emailed to you when you purchased Satellite Forms.
- A detailed description of the problem, including the steps you took leading up to it, and any error messages you received.
- The operating system version and language of the development computer on which you installed Satellite Forms MobileApp Designer.
- If you installed from CD, the Satellite Forms MobileApp Designer CD.

Chapter 1
What's New

If you are already familiar with Satellite Forms, read this chapter to learn about new features and improvements added to Satellite Forms in version 8.

What's New in Satellite Forms Version 8

Satellite Forms 8 introduces several exciting and powerful new features since the previous release, and also includes several important bug fixes. This includes the addition of new script language keywords, new extensions and sample projects, and updated documentation. The new and improved features are highlighted below.

New and enhanced features in version 8:

1 New capability to add color to forms and controls! With the new Colorizer extension, you can set form and control colors to enhance the asthetic appeal of your application, or to convey information with more impact. You can set a color theme to use throughout your application, modify colors on a form-by-form basis, or even change colors while a form is displayed. The Colorizer extension works on both the Windows Mobile and Palm OS platforms.

2 New integrated runtime engine capability! When you build your project, the Satellite Forms runtime engine and launcher icon are merged together into an integrated runtime engine EXE for Windows Mobile or PRC file for Palm OS. You no longer need to bundle the main runtime engine executable with your app as a separate file, because the launcher icon and runtime engine become one and the same. See Creating and Assigning a launcher icon image for your application for complete details.

3 New capability to display a splashscreen image while your app is loading! Now you can have your own color image or logo displayed while the application loads, instead of the traditional "Loading…" message. See Creating a splash screen for full details.

4 New Tables Search script function makes it easy to find records in a table that match a specific field value, with a minimum of scripting. This function carries out a linear search by looping through the table to find the first record where the field value matches the search term.

5 New Tables Lookup script function makes it simple to look up the value of one field in the first record of a table where another field matches the search term. With

this new script function you could easily search a table for a customer ID value, and obtain the corresponding customer's name, all with a single function call.

6 New Tables Backup script function allows you to copy a data table or tables to a backup location at any time, to safeguard against data loss, or to share data with another application. On the Palm OS platform, tables may be backed up to a memory card, while on the Windows Mobile platform tables can be backed up to any folder.

7 New GetAppName, GetAppCreator, and GetAppVersion script functions allow you to query application info at runtime. This makes it easier to adapt scripts to handle filenames that vary based on the app's creatorID, name, or version number.

8 New native barcode scanner control for Honeywell scanners on the Windows Mobile platform. The HoneywellScan control enables you to integrate barcode support into your app for scanners such as the Honeywell Dolphin 7600.

9 New native barcode scanner control for Datalogic scanners on the Windows Mobile platform. The DatalogicScan control enables you to integrate barcode support into your app for scanners such as the Datalogic Memor.

10 New FindFiles extension for the Windows Mobile and Palm OS platforms enables you to search for files and folders, and get information about them such as file sizes and dates. This extension makes it easy to generate a list of PDB files in the app's folder, or a list of documents in the My Documents folder, etc..

11 New HyperLink control extension for Windows Mobile and Palm OS makes it simple to add colored, underlined text links that respond to pen taps, just like a hyperlink in a web browser. You can easily change the link text, color, font, and location at runtime, and hide or show them as needed.

12 The ConnectionMgr extension for Windows Mobile has been enhanced to include new functions that obtain a list of available connections on the device, and to initiate a specific connection rather than just asking for the system to decide which connection to use. This makes it easier to launch a GPRS connection instead of Wi-Fi, for example.

13 When you build an application, any extensions used by the application, as well as the runtime engine SDDI DLL for Windows Mobile, are copied into the application target folder along with the other application and table files. This simplifies the app deployment process by placing all of the app files together in the target folder. When you use the Download App & Tables function from MobileApp Designer, the Windows Mobile SDDI DLL is downloaded along with the other app files. If you use the integrated runtime engine option, then your app folder will contain all files needed to run your app, with no need to pre-install the Satellite Forms runtime engine on the device.

14 New functions have been added to the SysUtils extension including an `SU_TapScreen` function for Palm OS and Windows Mobile, and `SU_PlaySoundFile` & SU_HideStartIcon functions for Windows Mobile. `SU_PlaySoundFile` enables you to play WAV audio files.

15 Added a new `CalcTextWidth` function to Color Graphics extension to get width in pixels of given string and font, for Palm OS and Windows Mobile.

16 Added a new SetBarExtra function to the Symbol Integrated Scanner control, used to control barcode scanning on Symbol and Janam barcode scanners. The SetBarExtra function allows you to enable or disable extended barcode symbologies that were not previously supported, including newer 2D symbologies like Maxicode, QR Code, and Data Matrix.

17 Added a new CalcDistance function to the Windows Mobile GPS extension to calculate the distance between two waypoints. The distance is returned in metres, and the waypoint coordinates must be specified in decimal degress, the same format returned by the GetPosLatitude and GetPosLongitude functions.

18 Added a new PocketPC target to the GoogleMaps sample project, demonstrating how to integrate your Satellite Forms Windows Mobile app with the Google Maps for Windows Mobile app to display locations, search businesses, and plot routes.

Bugs corrected in Satellite Forms 8:

1 The Windows Mobile runtime engine About dialog box now includes a standard OK button on the dialog, to make it easier to dismiss the About screen on Windows CE devices that do not display a standard OK button in the upper right corner like Windows Mobile does.

2 Under certain circumstances, multiple sequential calls to the PromptCustom function on a form could cause the app to crash with a stack overflow. This problem has now been corrected. [SF-00369]

3 The Satellite Forms help manual documentation for the Tables BinarySearch function has been improved to state that the RowNum variable will receive the correct sort position for the record, even if a match is not found in the table. This lets you know where a record should be located in the table in order to maintain the correct sort order.

4 The ConnectionMgr extension for Windows Mobile has been redesigned so that it no longer crashes upon load on Windows CE devices. The Windows Mobile platform includes ConnectionMgr API functions, but Windows CE does not. A new function lets you query the device to see if the ConnectionMgr is supported or not, befopre attempting to call any connection functions.

5 The ShowImage control for Windows Mobile has been redesigned so that it no longer crashes upon load on Windows CE devices. The Windows Mobile platform includes image decoding API functions used by ShowImage, but Windows CE does not. A new function lets you query the device to see if the extension is supported or not, befopre attempting to call any display functions.

6 On Palm OS platform devices, the SDK runtime list of applications now responds to the Enter key to select a desired app to run, instead of requiring a tap on the screen.

7 Tables with more than 128 columns did not display properly in the listbox on Windows Mobile. This is now fixed. [SF-00342]

8 A bug was corrected in the ShowImage control for Windows Mobile, where the control sometimes required two taps before executing the OnClick action. [SF-00390]

9 The SysUtils SU_LaunchAppAtTime function for Windows Mobile was updated to properly cancel a pending event by passing a time value of 0.

10 The runtime engine for Windows Mobile would display table-related error messages with an incorrect table filename, now fixed to show the correct table PDB filename.

11 The SU_RegReadKey function for Windows Mobile in the SysUtils extension had a memory leak, where the key value string memory was not properly released, now fixed.

12 The FormatDate function of the Strings extension for Pocket PC was updated to correctly handle the format string tokens “y” (day of the year), “q” (quarter), and “w” (day of the week as number). The output now matches the Palm OS version.

Other changes in Satellite Forms 8:

1 Satellite Forms support for the abandoned PocketPC CDB handheld database format has been removed. The CDB database format was abandoned by Microsoft several years ago, dating back to the introduction of Windows Mobile 5. Satellite Forms introduced support for the Palm PDB database format on the Windows Mobile platform starting with SatForms version 7, and as of version 8 the PDB database format is the only handheld database format supported by Satellite Forms on both the Palm OS and Windows Mobile platforms.

What's New in Satellite Forms V7.2

Satellite Forms 7.2 introduced numerous new features and bug fixes from the 7.1 release, highlighted below.

New and enhanced features since 7.1:

1 New Serial Port extension for Pocket PC provides Satellite Forms applications the ability to read & write serial port data on Pocket PC platforms (the Palm OS platform serial port extension was already available). The Pocket PC Serial Port extension supports a standard serial connection, raw SIR (serial over infrared), infrared IrCOMM, and Bluetooth RFCOMM (see the extension help in MobileApp Designer for details). The serial port sample project Terminal has been updated for Pocket PC support also (it includes both Palm OS and Pocket PC build targets).

2 New ShowImage control for Pocket PC provides Satellite Forms Pocket PC applications the ability to display common image files including JPG/GIF/PNG/BMP and more on the current form. Images are stretched/shrunk to fit the specified control rectangle. The ShowImage control can also act like a button control by handling pen taps if desired. A sample project (and sample GIF file) is included to demonstrate the control.

3 New ConnectionMgr extension is a plugin extension for Pocket PC that enables your application to initiate a dialup connection to the Internet. This is useful for TCPIP Winsock functions, HTTP, FTP, etc. on dialup TCPIP connections (eg. modem, EDGE/GPRS, 1xRTT/EVDO, HSDPA, etc.). A disconnect function is also provided.

4 New JanamUtils extension provides access to Janam XP20/XP30 hardware utility functions like toggling the keypad backlight, vibrator, LED, 5V Power Out, and Bluetooth power state.

5 New WM5Camera extension is a plugin extension for Pocket PC that enables your Windows Mobile 5 and higher applications to capture photos or videos on camera-equipped devices. Photos are saved to .JPG files in the folder you specify. Videos should be saved to .3GP files in order to be compatible with MMS standards among mobile phones.

6 A new GetAppPath application property was added, which returns a string containing the folder path in which the application resides on the Pocket PC device. This is useful for any instance in which you need to know the path to a file, for example the path to an image file used for the ShowImage control, or a BMP file created by the InkHelper extension, or another application for the SysUtils SU_LaunchApp function, etc. The use of the GetAppPath function to get the application's folder path enables your code that relies on files paths to keep working without modification even if your application is moved to a different folder (for example to an external memory card folder). So, you would not need to hardcode a folder path to files that are located in your app's folder or subfolders, instead you can now get that path using the GetAppPath method.

7 New IH_FileToBinField function in the InkHelper extension allows you to import a file (such as a BMP file) directly into a table binary field. This is useful when you want to access a file created on the handheld in your server database, with the file stored in a table field.

8 The Satellite Forms SDK runtime engine for Palm OS list of SDK applications now responds to up/down cursor keys, and the select button, to ease the selection of the SDK app to run.

9 The Pocket PC runtime engine now handles certain cursor key keypresses differently than before. Previously, in both edit and paragraph controls, the up/down keys would move the form to the previous or next record. Left/right would move to the previous/next page (or record for single page forms). Home/end would move to the first/last record. PgUp/PgDn had no effect. This is now changed to be more intuitive. In a paragraph control, the default is now to move the cursor within the paragraph instead of moving the form between records/pages. In an edit control the up and PgUp keys move to the previous page/record, while down and PgDn move to the next page/record. The left/right/home/end keys now move the cursor within the text in the edit control.

Bugs corrected in Satellite Forms 7.2:

10 Using the Int() and Int64() operators to convert a Boolean value (true or false) to an integer 0 or 1 was not successful on either the Palm OS or Pocket PC platforms. The value remained a Boolean. This is now corrected to behave as expected (converting a True value to 1 and a False value to 0). This bug was present all the way back to SatForms 3.0 from 1999, perhaps even earlier! [SF-00361]

11 A change in the Satellite Forms Pocket PC runtime engine version 7.0.1 (055) rendered some auto-repeating buttons inoperable. They would only work as standard single-tap buttons and no longer auto-repeated. This has been corrected [SF-00373].

12 Under certain conditions of filtered tables, the Pocket PC droplist control would not apply a filter properly when the filter was a part of the droplist control's action, and the droplist control was used as the snapshot for the filter criteria. This is commonly used in a "filtering multiple droplists" scenario as described in the

Linking Droplist Controls article in the Tips & Tricks section of the Satellite Forms guide. The Pocket PC droplist control has now been corrected to work properly in this regard like the Palm OS droplist control. [SF-00375]

13 In conjunction with the development of the Pocket PC ShowImage control extension (new in SatForms 7.2), we discovered that the Pocket PC runtime engine was making numerous numerous unnecessary screen redraws, causing screen flicker. These superfluous screen redraws are now suppressed. Another symptom of this issue was that under rare circumstances, "phantom controls" could be displayed on the form resulting a very jumbled display.

14 Support for reading and modifying the ReadOnly property of Pocket PC ink controls at runtime was added, to behave like Palm OS ink controls in this regard. You can now query and set the readonly property of the ink control at runtime. If you set the ink control readonly property to False, then the user cannot modify or clear the ink contents. [SF-00374]

15 The behaviour for the SF_DoButtonBehavior extension API on the Pocket PC platform was improved, to match the behaviour and appearance of that API on the Palm OS platform. You can notice this improved behaviour in the handling of pen taps on the "erase ink" box in the Pocket PC ink control.

16 The Pocket PC Strings extension was completely rewritten to eliminate string memory errors that were reported in previous versions. The previous string memory errors could lead to application crashes.

17 The Pocket PC Slider and Color Slider SFX controls have been updated to now move properly along with the rest of the control on the form when the form is scrolled (for example, when you pop up the onscreen keyboard).

18 The Pocket PC Color Graphics extension has been updated to correct some bugs. Problems corrected include (a) Drawing functions now work in color; and (b) a new SetPaintKeyCode function signals your app when the screen is redrawn, allowing you to repaint your graphics. The Color Table sample app now actually works as a result of these changes.

19 The runtime engine has been updated to work around a Palm OS NVFS (non-volatile file system) bug that would lead to "Cache_QueryRecord" errors in a Satellite Forms application. This NVFS bug is more prevalent on Palm OS devices that have Palm OS v 5.4.9 (for example the Palm TX or Janam XP30) compared to earlier OS releases. The bug is that the OS could sometimes leave database records marked as being "busy, in use" when they should not have been. This prevented the Satellite Forms runtime engine from accessing the table records properly, resulting in the error message. The runtime engine now attempts to clear the busy bit on those records to avoid the error.

20 The MobileApp Designer now limits the maximum number of global and local variables declared in an app to 255. Previously, MobileApp Designer would allow up to 1000 vars to be declared, but the Palm OS and Pocket PC runtime engines would only support up to 255 variables max. Increasing the declared variables over that limit would cause the application to crash. Increasing this limit beyond 255 vars requires changes to MobileApp Designer, the runtime engines, and the Satellite Forms conduit. This max var limit will be increased in a future version of Satellite Forms.

21 The Pocket PC ink control now properly ignores pen input when the ink control is not visible. Previously, the ink control responded even when invisible.

22 A string formatting problem in the Satellite Forms conduit could lead to "Invalid ArgsSize" and/or "Invalid ArgsSizePtr" error messages in the Palm OS Hotsync log, instead of correctly logging table transfers, on newer versions of Hotsync. This has now been corrected.

23 The RDKInst tool, as well as certain functions of the Satellite Forms HotSync ActiveX control (and utility DLL) would not function correctly with Palm OS HotSync 7.0.2, included with the Palm Desktop 6.2, and recommended by Palm for use on Windows Vista PCs. The user management functions in the SatForms utility DLL would no longer obtain the correct list of HotSync users, and as a result the list of users shown in the RDKInst tool would be blank. The HotSync user management functions in the SatForms utility DLL have been rewritten with the help of an additional new SFUsrIns.dll that is installed to the \Windows\System32 folder along with the other conduit DLLs. The RDKInst, DLL and ActiveX functions now work correctly with Palm Desktop 6.2 and HotSync 7.0.2, as well as previous versions. An additional aspect of this change is that user management changes are now effective immediately, and the HsCommitChanges/ HsAbandonChanges methods are now ignored.

24 Several of the Satellite Forms executables are now signed with an Authenticode digital certificate to prevent "unknown application" popups on Windows Vista. This includes the MobileApp Designer exe, RDKInst, CeRdkInst, and SFConvertPDB.exe. Previously, just the Satellite Forms installer was signed.

25 The default installation directory for Satellite Forms is now C:\Satellite Forms 7\ instead of C:\Program Files\Satellite Forms 7\ in order to improve compatibility with Windows Vista.

What's New in Satellite Forms V7.1

Satellite Forms 7.1 includes several important bug fixes, and also introduces numerous exciting and powerful new features since the 7.0 release. This includes the addition of new MobileApp Designer features, new script language keywords, new extensions and sample projects. The documentation has also been improved, with key sections updated and rewritten, and more than 40 new pages added. The new and improved features are highlighted below.

New and enhanced features since 7.0:

1 New CommitData script method added to the Table object, used to commit cached table data to storage on command. This **Tables(TableName).CommitData** method replaces the Palm OS DmSyncDatabase extension, and the Pocket PC CommitDatabase extension.

2 New NoAutoCommit table property and improved support of the Read-Only table property on Pocket PC enables Pocket PC PDB applications to close faster. Support for new NoAutoCommit property added to Mobile App Designer table editor, and to the SFConvertPDB utility.

3 New feature in MobileApp Designer to automatically Create an application shortcut (.Lnk) file when the Pocket PC application EXE is compiled. The shortcut

file is created in the AppPkg folder next to the EXE, and aids in application deployment.

4 New SysUtils extension provides access to dozens of useful operating system utility functions, including reading & writing to the Windows Mobile Registry and Palm OS Preferences, obtaining the device unique ID and model information, working with the system clipboard, parsing delimited text, and more.

5 New InkHelper extension provides utility functions for working with Satellite Forms Ink fields. It enables you to convert the ink data such as signatures and sketches into other formats including a BMP file, for easy integration with other software, on-device printing, and more.

6 New GPS extension provides easy access to GPS data on Windows Mobile 5 and higher devices, via the Windows Mobile GPS API.

7 New GoogleMaps extension for Palm OS enables your Satellite Forms application to launch the Google Maps application to find a location, find a business, get directions to a location, or get directions from a location.

8 New ScreenSize extension gives you information about the current screen size and orientation on the PocketPC device.

9 New IntermecScan control extension enables your application to control the integrated barcode scanner on many Intermec industrial Pocket PC devices.

10 Improved the compatibility of the Symbol Integrated Scanner control for Pocket PC by preventing the control from crashing on non-Symbol units.

11 Improved the UnitechScan control by adding compatibility with the Unitech PA550, PA650, and PA962 scanners.

12 Updated the SocketScan control for Palm OS to improve scanning speed and add compatibility with new laser SD scan card models.

13 Improved the Aceeca IDVERIFI Bar Code extension to make it compatible with the Aceeca Meazura scan wedge utility.

14 Improved the DynamicInputArea control to add support for controlling the DIA keyboard input pinlet.

15 Added new restore workspace when project loaded option to the MobileApp Designer preferences.

16 Added a Menu Sample project to demonstrate the use of custom application menubars.

17 Added a PocketPC redistribution sample for the Advanced Calculator and Work Order sample applications.

Documentation and feature changes since 7.0:

18 Revised the Deploying Pocket PC applications instructions to reflect improved deployment procedures for Pocket PC PDB applications.

19 Added documentation for the SFConvertPDB Utility which was previously documented in the Satellite Forms KnowledgeBase only.

20 Revised the Satellite Forms Synchronization for Pocket PC section with the updated recommendations for Pocket PC PDB applications.

21 Added or expanded descriptions for several Satellite Forms extensions.

22 Added dozens of new KnowledgeBase articles including both Known Issues and How Tos.

23 Revised the Quick Tour chapter to build a Pocket PC version of the Customers To Visit starter project, rather than a Palm OS version. The revised Quick Tour also includes instructions on how to make a Palm OS version of the sample project.

24 Added a new platform target named simply “PocketPC”. The PocketPC platform target is the recommended platform target to use for all PocketPC application development, and is identical to the PocketPC PDB platform target except for the simplified name. The older PocketPC platform targets are still included to provide backward compatibility with existing projects, but for all new development we recommend using just the “PocketPC” target.

25 Modified all sample projects to remove the PocketPC2003 build targets, and replace the PocketPC PDB build targets with the new PocketPC build target.

26 Amalgamated the SatSyncPPC and SatSyncPCPDB sample desktop sync applications into a single SatSyncPPC application. The revised SatSyncPPC sample demonstrates how to synchronize PocketPC PDB databases using both the Satellite Forms ActiveSync ActiveX Control method, and the non-ActiveX CeRemote.dll method, and provides the option to select which method to use.

27 Modified the PocketPC runtime engine installer EXEs and CAB files so that the main SatFormsRuntime_Install_RDK.exe (and CAB) contains support for PDB databases only, while the new SatFormsRuntime_Install_RDK_CDB.exe (and CAB) contains support for both PDBs and the obsolete CDB database format. The SatFormsRuntime_Install_RDK_PPCPDB.exe (and CAB) was removed.

28 Added menu shortcuts to the SatSync and SatSyncPPC sample desktop sync applications.

Bugs corrected in Satellite Forms 7.1:

29 A bug was detected in Pocket PC PDB record deletion behaviour, in which the data from deleted records would still be stored in the PDB file after the record was marked as deleted, instead of being properly removed. This has now been corrected. [SF-00365]

30 A couple bugs were detected in the behaviour of the droplist control for both the Palm OS and Pocket PC platforms, now corrected. [SF-00360, SF-00334]

31 MobileApp Designer exhibited a bug in the table editor, whereby new tables could be initialized with incorrect table properties. On Pocket PC PDB applications, the incorrect table properties could cause the table to not save data. Now corrected to initialize new table properties correctly. [SF-00352]

32 A bug was uncovered in the way that MobileApp Designer packaged high density bitmaps into the Palm OS application resource file, leading to application crashes on the handheld. Now corrected.

33 Under certain circumstances (including script code that writes a value to a table other than the form's linked table), the form's current table record pointer could get "out of sync" when deleting a record using the Pocket PC PDB database format. Now corrected. [SF-00347]

34 Updated the Pocket PC runtime engine to behave the same as the Palm OS when the form's current table record is filtered out of view. [SF-00340]

35 Sometimes Pocket PC keyboard input to the form could be ignored (with an error beep) until the form was tapped at least once. This is now corrected. [SF-00351]

36 Sometimes the Pocket PC form close X/OK button could disappear, if you switched away from the SatForms app to something else, and then switched back to the SatForms app. This has now been corrected.

37 Form vertical scrollbars should automatically appear and remain on Pocket PC applications that are running on display sizes that do not match the application target (for example running a standard 240x320 application on a device with a 240x240 square screen.

38 Depending on the length of text, presence of embedded linefeeds, and so on, in some circumstances prompt text and/or button text could be truncated in the PromptCustom dialog box on Pocket PC devices. This is now corrected. [SF-00278][SF-00355]

39 If a SatForms Pocket PC application was running and the user switched away to another app (like the Today screen), and then tapped on the SatForms app's icon a second time, the currently running app would correctly be brought to the foreground. However, after the app was closed, it would immediately restart again. This has now been corrected so that the app just gets brought to the foreground and does not restart again after it is closed. [SF-00329]

40 A bug in the WinMobile5Square platform definition file prevented the use of alternate shape radio buttons and checkboxes. This has been corrected.

41 A bug in all Windows Mobile/PocketPC platform definition files caused App Designer to always change the default desktop database format to MDB for those platform targets, when the project was loaded. This has been corrected. [SF-00327]

42 The SFConvertPDB utility used an incorrect (reversed) default CreatorID if the CreatorID parameter was not supplied on the commandline. It was incorrectly using FSMS and has been corrected to use SMSF as the default.

43 Implemented alternate-shape button control on PocketPC as a button with an autoresizing caption unlike the nonresizing caption of the standard button control. This matches the autoresizing caption behaviour of the altshape button on the Palm OS platform.

44 Pocket PC PDB Tables().Binarysearch function returns an incorrect sort position row number when the search value is not found. It returns a sort position row number 1 higher than it should. Now fixed. [SF-00328]

45 Pocket PC PDB sort and binarysearch case sensitivity behaviour has been updated to match the behaviour of the PalmOS runtime engine. [SF-00338]

46 Pocket PC PDB cannot filter records on numeric fields with width >= 10 digits, resulting in incorrect filtered recordset. Cannot display/edit numeric table fields with width >= 10 correctly, resulting in invalid data warnings. Now corrected to display and filter numeric fields wider than 9 digits correctly. [SF-00339]

47 The CTRL-F hotkey in MobileApp Designer was incorrectly performing a Find Next function, and has been corrected to perform the expected Find function.

What's New in Satellite Forms V7.0

Satellite Forms 7.0 introduces several exciting and powerful new features, building on the previous 6.1.1 release. This includes the addition of new control properties, new script language keywords, new extensions and sample projects. Version 7.0 delivers improved database performance and synchronization for the PocketPC platform, and improved compiler speed in App Designer, plus many more improvements listed below.

New and enhanced features since 6.1.1:

1. PocketPC applications can now use the more efficient Palm Database (PDB) format for handheld devices tables, in addition to the Microsoft Pocket Access Compact Database (CDB) format that was supported previously. The use of PalmDB format tables on the PocketPC platform offers numerous advantages, including:

- PDB tables are significantly smaller than CDB tables because of their more efficient structure. While CDB tables have a minimum size of 48KB even when they are empty (128KB on PocketPC 2002 devices), the minimum size of a PDB table is a mere 80 bytes.
- Using PDB tables on the PocketPC removes the reliance on the Microsoft ActiveSync functions that convert desktop databases to/from the handheld CDB databases, thus improving the speed and reliability of data synchronization. With the changes Microsoft made to ActiveSync 4.x, handheld database synchronization using CDB tables has become less reliable than it was with ActiveSync 3.x, especially on Windows Mobile 5 powered devices. Using PDB tables on the PocketPC avoids these reliability and performance issues.
- With the use of PDB tables on the PocketPC and PalmOS devices, the data tables can be transferred between platforms with complete compatibility. You can use the same PDB tables created on the desktop PC with both PocketPC and PalmOS handhelds. You can transfer PDB tables directly between devices on different platforms, using infrared beaming, Bluetooth, or SD memory cards.
- Compiling Satellite Forms applications for the PocketPC platform is quicker and easier using PDB tables than it is with CDB tables, because there is no need to generate the handheld tables on the connected device at compile time (this is how CDB tables are created, which makes the compile process take longer). With the PDB format, the data tables can be generated directly on the desktop PC without needing a connected PocketPC device.
- The same application target can be used for PocketPC 2002 devices in addition to PocketPC 2003, 2003SE, and Windows Mobile 5 for PocketPC devices. There is no need to build a separate application target for PocketPC 2002 devices when using PDB tables, as there was when using CDB tables. PocketPC 2002 CDB tables are not compatible with PocketPC 2003 and later devices.

An existing PocketPC project may be changed in the Project Properties to use the PalmDB format device databases instead of PocketPC DB. Also, a new PocketPC PDB target platform has been added, which you can add to existing projects or use for new projects, which uses the PalmDB table format by default.

The sample projects included with Satellite Forms 7.0 have been updated to include new PocketPC PDB build targets, and to remove the PocketPC 2002 build targets.

See the new KnowledgeBase article 10033 "How To use PalmDB (PDB) tables in a PocketPC application" for more information.

2. Satellite Forms applications can now utilize the expanded screen sizes available on some PalmOS devices, such as the Palm Tungsten T3, T5, LifeDrive, and TX. A new DynamicInputArea SFX control is provided that enables you to control the behaviour of the dynamic input area (DIA, also known as soft Graffiti area or collapsible Graffiti area) and screen rotation features. A sample project called DynamicInputArea is provided to demonstrate this capability. As well, new platform targets called Palm Tall and Palm Wide have been added, to aid in the creation of applications that are designed specifically to use the taller or wider screen sizes. See the new KnowledgeBase article 10036 "How To support Expandable Screens in PalmOS applications" for more information.

3. PalmOS applications can now support high density (hi res) bitmaps and icons. Most PalmOS 5.x devices support high density displays that offer 4 times the resolution of the standard 160x160 pixel displays on older PalmOS devices. Your SatForms applications can now take advantage of this by including higher quality 8-bit and 16-bit high density (HD) bitmaps, and high density launcher icons.

To add an 8-bit HD bitmap, name the bitmap with the -8-HD.bmp suffix and include it in your Images folder alongside the standard density bitmaps. To add a 16-bit HD bitmap, name the bitmap with the -16-HD.bmp suffix.

New large and small HD icon templates have also been provided.

Note that HD images take 4 times the number of pixels as standard density images, thus you will need to be more careful to not exceed the 64KB maximum size limit of a bitmap family.

See the new KnowledgeBase articles 10034 "How To use High Density Bitmaps in PalmOS applications" and 10035 "How To use High Density Icons for your PalmOS applications" for more information.

4. A new UnitechScan extension has been added to control the integrated barcode scanner on Unitech PocketPC scanner devices, such as the PA-950 model. See the sample application "UnitechScan" for more information.

5. A new LaunchURL extension for PalmOS and PocketPC has been added as a standard Satellite Forms extension. See the Satellite Forms Solutions Guide topic "Developer Tools | Satellite Forms Extensions | Network & Email Tools | LaunchURL" for more information.

6. A new LaunchReturn extension for PalmOS 5.x devices has been added as a standard Satellite Forms extension. See the Satellite Forms Solutions Guide topic "Developer Tools | Satellite Forms Extensions | General Purpose | LaunchReturn" for more information.

7. New .SetPosition and .GetPosition methods have been added for Satellite Forms form controls, enabling you to determine and modify the location and size of controls on the form at runtime. This capability complements the new DynamicInputArea support on PalmOS devices, and is available on both the PalmOS and PocketPC platforms. The x, y, width, and height of controls can be queried and modified at runtime. See the new KnowledgeBase article 10037 "How To Move and Resize Controls at Runtime" for more information. A sample project titled "DynamicInputArea" is provided.

As a result of this addition, the SetPosition/GetPosition/SetMinMax functions of the Slider and Color Slider controls have been renamed to SldSetPosition/SldGetPosition/SldSetMinMax to prevent function name conflicts. You will need to use the Find & Replace function to change these functions in your applications in order to make them compile properly in SatForms 7. See the KnowledgeBase article 10040 "Error C016: Method "Controls.SetPosition" takes 4 param(s) : 1 specified" for the full details. Note that this also affects users of the third party LSListBox extension.

8. New .Popup method has been added for droplist controls, to pop up the droplist list via script command, and for edit & paragraph controls to pop up an automatic keyboard for the edit/paragraph control if one was defined for that control at design time.

9. New .Font method has been added for Satellite Forms form controls, enabling you to determine and modify the font used by a control at runtime. This capability is available on both the PalmOS and PocketPC platforms. See the new KnowledgeBase article 10041 "How To Change Control Fonts at Runtime" for more information.

10. SFConvertPDB is a new commandline utility included with Satellite Forms 7.0 that enables you to convert database files from PDB format to/from DBF or MDB format, on the desktop PC. It does not require a Palm HotSync or Microsoft ActiveSync session to be active, as it runs entirely on the PC. This provides a handheld-platform-independent PC based mechanism to convert data to & from PDB files, for use with Satellite Forms applications on the PalmOS platform and on the PocketPC platform when using PalmDB database tables. SFConvertPDB is intended for use both as a database conversion component of a synchronization system, and also as a developer data conversion utility. See the new KnowledgeBase article 10038 "How To use the SFConvertPDB utility" for the complete details.

11. A new sample synchronization tool and developer utility called SatSyncPPCPDB is included. SatSyncPPCPDB demonstrates how to send and receive data between the desktop PC and PocketPC handheld, using the Satellite Forms CeRemote.dll and SFConvertPDB utility. This sample is designed for PocketPC applications using PalmDB (PDB) format databases on the handheld. SatSyncPPCPDB is supplied in Visual Basic 6 source code form, and also a compiled setup package ready to use. See the new KnowledgeBase article 10039 "How To use SatSyncPPCPDB to sync PocketPC data" for the full details.

12. A new BatteryInfo extension for PalmOS and PocketPC has been added as a standard Satellite Forms extension. See the sample project "Battery Info" for more information. The source code for this extension is also provided in the \Samples\Extensions\BatteryInfo folder.

13. Bachmann PrintBoy extensions and sample projects have been added to Satellite Forms. This includes the HTML Edition extension for PocketPC, and the Basic, Reports, and HTML Editions for PalmOS.

14 A new FormNavHelper extension for PalmOS 5.x has been added as a standard Satellite Forms extension. See the sample project "FormNavHelper" for more information.

15 CeRemote.lib and CeRemoteAPI.h library files have bee added to document the public functions of the CeRemote.dll for transferring files between the desktop PC and PocketPC. The use of the CeRemote.dll for PocketPC synchronization is demonstrated in the new SatSyncPPCPDB sample project/tool. For more

information, see the KnowledgeBase article 10039 "How To use SatSyncPPCPDB to sync PocketPC data".

16. The performance of the application compiler in App Designer has been dramatically improved for complex applications that make extensive use of global functions and subroutines, reducing the amount of time it takes to compile your project. Some beta testers have reported speed gains in excess of 18000% compared to Satellite Forms 6.x!

Bugs corrected in Satellite Forms 7.0

17. The Deliveries sample project was updated to correctly load the embedded images for the customer maps and delivery items. This demonstrates the HSBM method of embedding images into databases (see the KnowledgeBase article 10010 "How To use color bitmaps in your application" for more information about the HSBM bitmap method).

18. The Deliveries sample synchronization app now includes support for both PalmOS and PocketPC synchronization (using the SF ActiveSync OCX and CDB files).

19. The PDF toolbar icon in App Designer to open the Satellite Forms manual in PDF format was opening the Satellite Forms Solutions Guide by mistake: this has now been corrected.

20. Previously, App Designer would complain about missing DLL files if you opened a project that included PocketPC build targets, but your development PC did not have Microsoft ActiveSync installed. This warning message was harmless but confusing, and has been removed.

21. An error with the Round function of the Math extension for PocketPC has been corrected.

22. The default height of a droplist control in App Designer for PocketPC targets has been increased from 17 to 19 pixels, to match the actual height of the droplist control on the PocketPC device.

Chapter 2
Satellite Forms Overview

Read this chapter to learn about:

- Overview of Satellite Forms
- Major components of Satellite Forms
- Basic information on relational databases and Satellite Forms
- Tables and how they are used in Satellite Forms
- Forms and how they are used in Satellite Forms
- Controls and how they are used in Satellite Forms
- Scripting in Satellite Forms
- Extensions in Satellite Forms
- Targets and Platforms in Satellite Forms

Introducing Satellite Forms

Satellite Forms™ is an integrated software development environment (IDE) that makes it easy to create custom applications for Palm OS and Windows Mobile/Pocket PC devices. Satellite Forms allows you to create usable, real-world applications without writing a single line of code. Even more sophisticated applications require minimal scripting or coding to implement.

Using Satellite Forms, you can create mobile applications that access valuable information from company databases – for example, customer orders or contact information. You can design applications to be read-only or to allow users to add or update data that can then be transferred back to a company database.

A desktop application you create directs how Satellite Forms applications and data are processed by Microsoft ActiveSync for Pocket PC devices, or Palm HotSync Manager for Palm OS devices. This means you have complete control over downloading data to the device and reintegrating the data back into a database.

You can use Satellite Forms to create applications for many tasks, including the following:

- Sales Automation
- Marketing Research

- Patient Records
- Inventory control
- Pharmaceutical Detailing
- Inspections
- Collections
- Repair Service Reports

The Satellite Forms applications you create provide several benefits to your end users and to your company or client. They make important and useful database information portable and easy to access. They make data entry and verification fast and accurate. They reduce costs by cutting down on paperwork and eliminating the need for re-keying information.

Major components of Satellite Forms

Satellite Forms has four main components:

- **MobileApp Designer:** The integrated development environment (IDE) you use to design the forms and tables of your applications. MobileApp Designer allows you to create applications with multiple forms and tables.
- The **Satellite Forms PalmOS Conduit:** Manages the transfer of data to and from PalmOS handheld devices and the desktop. (Not applicable to the Pocket PC platform.)
- The **Satellite Forms ActiveX controls:** Work with HotSync or ActiveSync technology to simplify integration with database applications.
- The **Satellite Forms Engines:** Run Satellite Forms applications on PalmOS and Pocket PC handhelds.

Satellite Forms uses the industry standard dBase™ and Microsoft® Access™ formats to transfer information between desktop and handheld devices, making the applications you create using Satellite Forms compatible with the leading database products.

Licensing Satellite Forms

Satellite Forms is licensed on a per-developer (individual person) basis. A single developer may install and use Satellite Forms on up to two (2) PCs at one time (for example one desktop PC and one laptop), but each individual developer requires a separate license. The handheld applications and conduits you create with Satellite Forms are royalty-free.

You can utilize Satellite Forms to create an unlimited number of applications, distributed to an unlimited number of handheld users, with no royalty fees.

You are free to distribute the runtime components and RDK engine (SF80RDK.prc), SDDI plug-ins, SF Runtime Installer for Pocket PC and its content, including the SatForms80.exe and DvSDDI_PPCPDB.dll, and extensions for Satellite Forms applications that you develop with MobileApp Designer.

The SDK engine (SF80SDK.prc) is not redistributable. You are not permitted to install the Satellite Forms development package on end-user computers (unless you purchase a developer license for each end user), but you are permitted to install/deploy the runtime components.

- Email: Sales@SatelliteForms.net
- Other contact information: www.SatelliteForms.net

Relational databases and Satellite Forms

One of the most common methods of organizing data on computers is the relational database. Database management systems (DBMS) such as Microsoft Access are applications that use the relational model to store and retrieve information.

With any DBMS, data is stored and presented to the user using a distinct method. Data is stored in one type of object, usually called a table. Data is displayed in another type of object, usually called a form.

Any application that uses the relational model – Satellite Forms included – consists of a collection of tables. The forms are the view windows or screens that present the data to users. Forms display information through controls placed and configured when the form is created. Satellite Forms applications use tables, forms, and controls in much the same way as any DBMS.

How Satellite Forms uses tables

Like other relational databases, Satellite Forms uses tables to store and retrieve information. An application created with MobileApp Designer can use one or many tables.

The following figure shows a sample Clients table:

Figure 2.1 Clients table

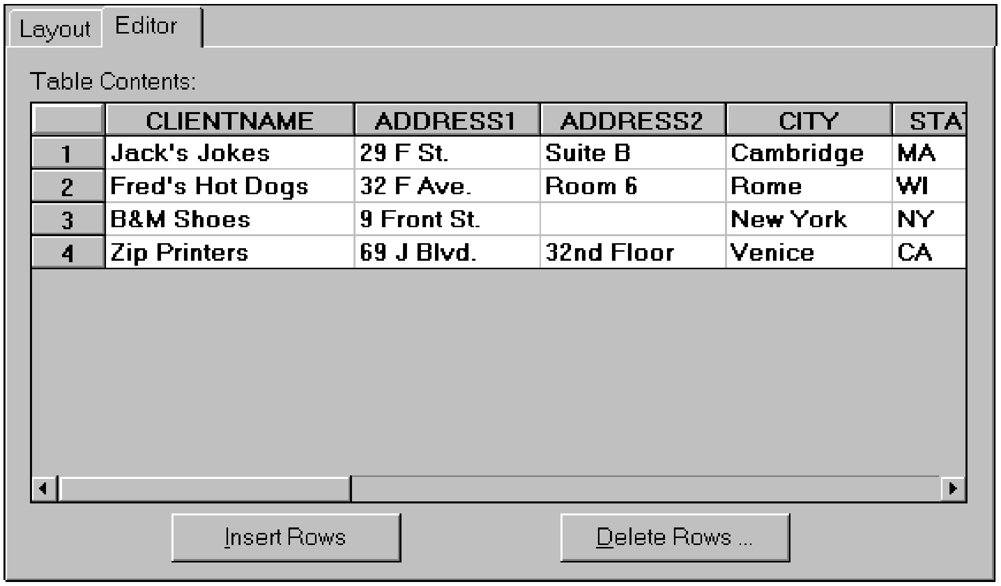


Table Contents:

	CLIENTNAME	ADDRESS1	ADDRESS2	CITY	STA
1	Jack's Jokes	29 F St.	Suite B	Cambridge	MA
2	Fred's Hot Dogs	32 F Ave.	Room 6	Rome	WI
3	B&M Shoes	9 Front St.		New York	NY
4	Zip Printers	69 J Blvd.	32nd Floor	Venice	CA

The table has four client records, each with seven fields, only five of which are shown in the example: CLIENTNAME, ADDRESS1, ADDRESS2, CITY, STATE, ZIPCODE, and PHONENUM.

This example illustrates several important table requirements:

- Each table in a Satellite Forms application must have a unique name.
- Each table has rows and columns – that is, records and fields.
- Each field is labeled with a field name describing the information stored in the field (for example, CLIENTNAME).
- Each field is assigned a specific data type that determines the characters and format the field accepts. For example, only numbers can be stored in a numeric field.

Satellite Forms tables support the following field data types:

- **Character:** Accepts any printable character: letters, numbers, and symbols.
- **Numeric:** Accepts numbers only, such as prices, inventory quantities, ID numbers, and so on. Note that phone numbers, zip codes, and numbers representing dates and times should not be entered into numeric fields. Use the Character data type for phone numbers and zip codes. Dates and times have their own corresponding data types.
- **True/False:** Accepts only T for true or F for false. Use for Boolean data.
- **Date:** Accepts only date information in MM/DD/YYYY format in MobileApp Designer. On the handheld device, dates are displayed and entered in a country–specific format according to the **Preferences** settings on the handheld device.
- **Time:** Accepts only time information in the HH:MM am/pm format in MobileApp Designer. On the handheld, times are displayed and entered in a country–specific format according to the **Preferences** settings on the handheld device.
- **Binary:** Accepts input from an Ink control for signatures, drawings, and so on. This data appears on the desktop computer as an OLE object.
- **Time Stamp:** Accepts an eight-bit integer used to establish compatibility with imported Oracle Lite tables.

In the Clients table, as shown in Figure 2.1 on page 21, all fields are defined to accept Character data. The ZIPCODE and PHONENUM fields are not numeric because they contain dashes, parentheses, or both.

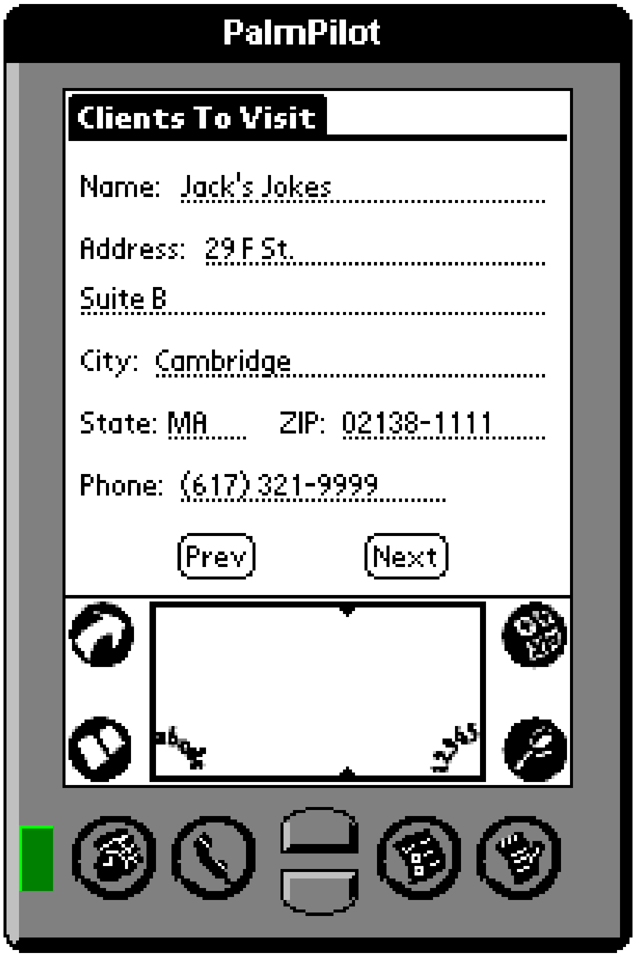
In the Clients table, each record contains information for an individual company. Each field contains a particular type of data, for example, a company name or address. For an individual company, a particular field may be empty. In the Clients table, B&M Shoes has no information in the field ADDRESS2 because there is no suite, room, or floor number for that company.

How Satellite Forms uses forms

Satellite Forms uses forms to display information. The forms you create using MobileApp Designer become the screens that appear on the handheld device when an application is running. Just as each application can use multiple tables, each application can also have multiple forms.

The information a form presents may be contained in the form itself as static text, as in the case with a simple Help form, or can be retrieved from a database table. Each form you create using MobileApp Designer can have a table linked to it. When you link a form to a table, the form's controls display the current record of that table. For example, a form linked to the Clients table, as shown in Figure 2.1 on page 21, might look like the one shown in the following figure:

Figure 2.2 Clients to Visit form



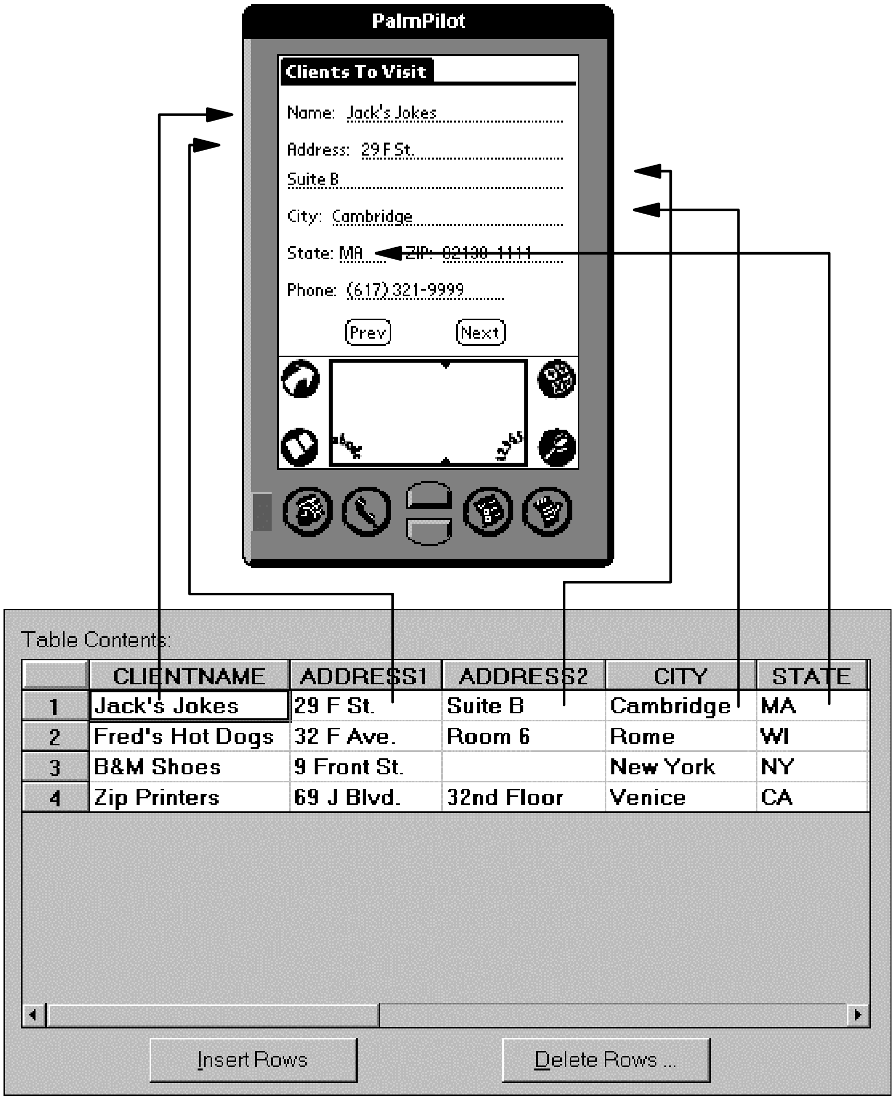
Name: Jack's Jokes
Address: 29 F St.
Suite B
City: Cambridge
State: MA ZIP: 02138-1111
Phone: (617) 321-9999

This Clients To Visit form shows the contents of the current record, Jack's Jokes, from the Clients table.

Each form can only be linked to one table. The form in Figure 2.2 is linked to the Clients table. Each item on this form is a control that displays static information or the contents of a particular field of the current record, or starts an action when you select it.

On the Clients to Visit form, the title is a static Text control. The buttons labeled Next and Prev are controls that perform an action, displaying the next or previous customer in the Clients table. The lines displaying information are controls linked to the data in the current record of the Clients table. Each of these controls has as its data source a specific field in the table. The following figure shows the Clients to Visit form, the Clients table, and the links between the form's controls and the table data sources:

Figure 2.3 Clients to Visit form and current record of linked Clients table



Clients To Visit

Name: Jack's Jokes

Address: 29 F St.

Suite B

City: Cambridge

State: MA ZIP: 02138-1111

Phone: (617) 321-9999

Table Contents:

	CLIENTNAME	ADDRESS1	ADDRESS2	CITY	STATE
1	Jack's Jokes	29 F St.	Suite B	Cambridge	MA
2	Fred's Hot Dogs	32 F Ave.	Room 6	Rome	WI
3	B&M Shoes	9 Front St.		New York	NY
4	Zip Printers	69 J Blvd.	32nd Floor	Venice	CA

When a form is initially displayed, or a new record is displayed, the controls are loaded with the data from their respective data sources. As you use the form and edit the data, changes are saved to the linked table only when you move to another record or leave the form.

Controls used in Satellite Forms

The controls in Satellite Forms determine what is displayed and what actions can be performed on any form created with MobileApp Designer. The types of controls available are described below:

- **Title control:** Displays a title across the top of the form, as shown in Figure 2.2 on page 23.

- **Text control:** Shows static text – that is, text that does not change when the application runs.
- **Edit control:** Displays and allows editing of the contents of a particular field of the current record of the form's linked table. The controls displaying the client information on Jack's Jokes, as shown in Figure 2.2 on page 23, are Edit controls. Each Edit control is bound to a field in the form's linked table. Edit controls were called Input Field controls in previous versions of Satellite Forms.
- **Paragraph control:** Like an Edit control, but displays multiple lines of information. Use Paragraph controls for entering or displaying notes or other lengthy, unformatted text.
- **Check Box control:** Represents an item that can be checked off. For example, you could use Check Box controls to create a form that surveys how a customer uses a software product – for entertainment, business, school, work, and so on. The customer checks any or all choices that apply. The Check Box control can only be bound to a True/False field in the form's linked table.
- **Radio Button control:** Used like the Check Box control, but only one choice can be selected within a group of Radio Button controls that are bound to the same field in the form's linked table. Use Radio Button controls for selections where multiple choices are not appropriate.
- **Button control:** Executes an action or a script, as illustrated by the Next and Prev buttons.
- **List Box control:** Displays the contents of a form's linked table in a tabular format. The control can display some or all of the table's fields. Use this control to display more than one record at a time or to group and display selected fields. For example, you could set up a List Box control to display all the customer names and telephone numbers from the Clients table at one time and not have to move from record to record to find the desired information.
- **Drop List control:** Displays a drop-down list, like a Windows combo box, from which the user can select an item. The contents of the Drop List control, as with the Lookup control, are derived from a lookup table, not the form's linked table.
- **Lookup control:** Displays data from a lookup table. Use this control to display information from a form's linked table in a more useful format. For example, you can display customers identified by account numbers by name, rather than account number.
- **Ink control:** Allows freehand drawing or signatures, which appear on the desktop computer or server as OLE objects. For example, use this control to record a signature on screen. The Ink control can only be bound to a Binary field in the form's linked table.
- **Bitmap control:** Displays a static bitmap. Use this control to place a company logo on a form or overlay an illustration on a Button control. You can also use it as a background to create more interesting and attractive forms.
- **Graffiti Shift Indicator control:** Places an indicator graphic on the handheld device's screen that shows the shift state – lowercase, uppercase, or caps lock – of the Graffiti handwriting recognizer.

Note This control is not available for Pocket PC applications.

- **Auto Stamp control:** Automatically enters a date or time stamp in a field. This control is invisible to the user.
- **SFX Custom control:** Some SFX Custom controls can be bound to a table field. The Slider control provided with Satellite Forms is an example of an SFX Custom control.

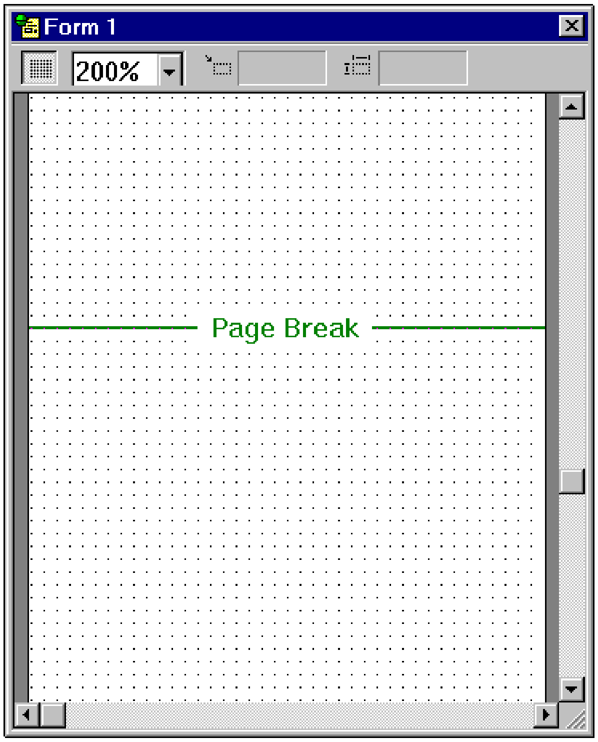
Multiple forms and pages

In Satellite Forms, each form can have only one linked table. When you need to use more than one table in an application, you must create a separate form to link to each table or use the scripting language to provide custom functionality.

When using multiple forms, a Button or other control that provides actions makes it possible to jump from the current form to the next or previous form. This allows an application created using MobileApp Designer to access information stored in as many tables as necessary. For more details on using multiple tables and forms, see Jump to Multiple Forms options on page 180.

A form can also have more than one page if the information to be displayed is more than one handheld computer screen can hold. A multi-page form is still a single form. All the pages of the form display data from the same record. An example of a form with more than one page is shown in the following figure:

Figure 2.4 Form with multiple pages



Tip Multiple single-page forms are easier to design and maintain than a single multiple-page form, especially if you use scripting. To the user, there is no apparent difference between the two approaches, so we recommend using multiple single-page forms for simpler design and maintenance.

Navigating between pages and records

The **Up** and **Down** scroll buttons on handheld devices move between records in a table. If a form has more than one page, clicking the **Up** or **Down** button moves between pages as well. If you click the **Up** button while viewing the first page on a form, the handheld device displays to the previous record in the form's linked table. If you click the **Down** button while viewing the first page on a form, the handheld device displays the next page on the form. While viewing the last page on a form, clicking the **Down** button displays the next record in the form's linked table. Clicking the **Down** button while viewing the last page of a form which is displaying the last record in the form's linked table can create a new record if the user permissions are set to allow it.

Scripting

Satellite Forms scripts allow you to add custom functionality of your handheld applications. The Satellite Forms scripting language uses syntax similar to Microsoft Visual Basic®. Scripting complements built-in Satellite Forms features with local and global variables, mathematical operations, conditional logic, loops, and user-interface functions. For example, with Satellite Forms scripting, you can:

- Perform arithmetic calculations and logical operations on data.
- Add business logic and complex validation to a form.
- Access the methods and properties of Satellite Forms objects.
- Manage data in tables and perform functions like cascade updates and cascade deletes.
- Create template applications that dynamically change based on user input.

For more information and examples on how to use the Satellite Forms scripting language, see Satellite Forms Scripting Language Reference, on page 257.

Extensions: SFX plug-ins and SFX Custom controls

The Satellite Forms Application Programming Interface (API) allows you to expand the capabilities of Satellite Forms with extensions called SFX plug-ins and SFX Custom controls. SFX plug-ins are non-visual, functional enhancements for your applications that expand the script language. SFX Custom controls are visual controls you can place on forms, and can also expand the script language. You can write your own extensions or use extensions written by others. For example, Satellite Forms provides several extensions, including an SFX plug-in called Square Root and an SFX Custom control called Slider. You can add these extensions to your applications or create new ones of your own design. The SFX plug-in and control extensions included with Satellite Forms, on page 268 lists many of the extensions included with Satellite Forms, many of which also have a sample project to demonstrate their features.

A categorized and searchable database of available Satellite Forms extensions is included in the **Satellite Forms Solutions Guide** in the \Satellite Forms 8\Doc folder, and is also available on the http://www.satelliteforms.net/ website.

Extensions are programs written in the C language – using Metrowerks CodeWarrior, for example – that exchange data with the Satellite Forms Engine on the handheld device. You can use extensions to manipulate data, perform complex business logic, create custom controls, display dialog boxes, and handle many other functions. For example, you can use extensions to:

- Extend the Satellite Forms scripting language with libraries of financial and statistical functions.
- Create new user-interface objects.
- Build drivers for devices such as barcode scanners, portable printers, and pagers.
- Add business logic and complex validation to a form.

For more information and examples on how to incorporate existing SFX plug-ins and SFX Custom controls into your applications, see Adding extensions to Satellite Forms on page 188. For more information on creating your own SFX plug-ins and SFX Custom controls, see Satellite Forms API Reference, on page 467.

Targets and Platforms

Satellite Forms has the ability to separate a single project into separate classifications called "Targets". A "Target" in MobileApp Designer is the current project's code and structure as applied towards a specific platform. This means that one application can be developed for both Palm OS and Pocket PC platforms. And, all the code will be saved in one project file. To add a new target to a project go to the "Build" menu and select "Targets".

All tables, forms, and controls are attached to the central project and can be shared between all the targets or excluded from specific targets. In order to exclude a specific form or control simply delete that control from within the desired target. If this object still exists in other targets it will appear greyed out in the "Project Contents" workspace, meaning it is no longer included in this target.

Properties can also be individually shared or excluded between projects as well. In the "Property Space" each checked checkbox on the left side of the window means that corresponding property is being shared amongst all targets. If you wish to modify a property for one target but not have that value transmitted to the other targets you must uncheck that property in the aforementioned target.

Global scripts are also separated into Shared and Private sections. Shared global scripts are available to all of the targets in a project, while Private global scripts are only available to the current target. This enables you to have different platform-specific scripts in each target, referenced by the same function or subroutine name.

With multiple target support, it's easy to take a project designed for one platform, and extend that application by building it for both the PalmOS and PocketPC platforms.

Chapter 3
Installing Satellite Forms

This chapter explains how to install Satellite Forms.

System requirements

The following sections define the hardware and software requirements for the Satellite Forms development computer and for the handheld devices on which you can install Satellite Forms applications.

Satellite Forms development computer

The following lists describes the recommended hardware and software minimum requirements for the computer on which the Satellite Forms development environment is installed.

Hardware requirements

The following list describes the minimum hardware requirements for the computer on which you develop Satellite Forms applications:

- PC with a Pentium-class processor, 200 MHz or faster
- 64 MB RAM
- 50 MB available disk space

Operating system software requirements

Satellite Forms has been tested with the following operating systems:

- Microsoft Windows Vista Business edition
- Microsoft Windows XP Professional edition
- Microsoft Windows 2000 Professional edition
- Microsoft Windows NT 4.0 Workstation with SP 6
- Windows ME
- Windows 98 Second Edition

Additional requirements

Satellite Forms requires the following additional software and configuration:

- To develop PalmOS applications: Palm HotSync version 3.01 or later.

  HotSync is installed with the Palm Desktop. The Palm Desktop software should be supplied with your Palm OS device, or may be downoaded from the device maker's website.

  To develop Windows Mobile/Pocket PC applications: Microsoft ActiveSync version 3.5 or later, or Windows Mobile Device Center for Windows Vista.

  You can download a free copy of ActiveSync or Windows Mobile Device Center from:

  www.microsoft.com/windowsmobile/activesync/default.mspx

- MDAC version 2.6, SP1

  The Microsoft® Data Access Components (MDAC) is software that Satellite Forms uses as the interface to its transfer tables. Earlier versions of MDAC are installed with Windows 2000 and Access 2000, but version 2.6 or higher is recommended for Satellite Forms.

  You do not need to update or install MDAC if any of the following conditions are true:

  - You are using Windows XP or Access 2002
  - You are using Access 97, in which case you can only use DBF format

  You need to update or install MDAC if all of the following are true:

  - You are using Microsoft Windows 98, Windows NT4.x, or Windows 2000
  - You are using Microsoft Access 2000

  If you update/install MDAC 2.6, you also need to update the Microsoft Jet 4.0 engine. For information, see www.microsoft.com/data/download.htm#Jet4SP3info

Note Installing MDAC may break other applications on the desktop computer. Please refer to the Microsoft articles listed below for details:

More information about MDAC can be found at www.microsoft.com/data

Useful articles on MDAC can be found on the following Microsoft sites:

support.microsoft.com/support/kb/articles/Q239/1/14.ASP

support.microsoft.com/support/kb/articles/Q230/1/25.ASP

Satellite Forms runtime engine

The following list describes the recommended hardware and software minimum requirements for the handheld devices on which the Satellite Forms runtime engine is installed.

- Palm OS devices
  - Palm OS 3.5 or later

  - 1 MB available main memory
  - HotSync 3.01 or higher
- Windows Mobile/Pocket PC devices, including these OS versions:
  - Pocket PC 2002, Windows Mobile/Pocket PC 2003, Windows Mobile/Pocket PC 2003 SE, Windows Mobile 5 for Pocket PC (including Phone Edition), Windows Mobile 6.x Classic, Windows Mobile 6.x Professional
  - *Note that Windows Mobile "Standard" editions do not support a touchscreen, and are not compatible with Satellite Forms.*
  - 10 MB available main memory
  - ActiveSync 3.5 or higher, or Windows Mobile Device Center

Upgrading from previous releases

If you are upgrading from an earlier version of Satellite Forms, you do not need to uninstall the previous version from your development PC. Satellite Forms 8 will co-exist with older versions of SatForms on the same PC. However, because Palm Hotsync only supports a single conduit to be registered for each unique creatorID, the conduit used by MobileApp Designer for the Download/Upload function will only be active for one specific version of SatForms. When the installation is complete, Hotsync will prompt you to select which conduit you wish to use: the SatForms 8.0 conduit, or the older version. Choose the newer SatForms 8 conduit in order to be able to use the Download/Upload feature from Satellite Forms 8 MobileApp Designer. Note that if you are installing SatForms 8 to a PC that already has an older version, make sure that you install it to a different folder: the old and new versions must not be installed in the same folder.

You do need to complete the following step before installing the new version:

- Uninstall the old Satellite Forms Engine for PocketPC from your handheld. For instructions, see Uninstall the Satellite Forms Engine from a Pocket PC device on page 37.

Installation overview

Installing Satellite Forms is a three-step process:

1. Install MobileApp Designer on your development computer
2. Install the Satellite Forms engine on your handheld device
3. Download and test a Satellite Forms application on your handheld device

Preparing for installation

Before installing MobileApp Designer, verify the following items:

- You have the license key. The license key was emailed to you when you purchased Satellite Forms.
- Your development computer meets the system requirements listed under Satellite Forms development computer on page 29.

- The required handheld device software is installed:
  - **Palm OS devices:** Install Palm Desktop
  - **Pocket PC devices:** Install Microsoft ActiveSync

Note If Microsoft ActiveSync or the Palm Desktop and HotSync Manager are not installed on your development computer before you install Satellite Forms, an error message appears if you attempt to install Satellite Forms.

Installing Satellite Forms on your computer

Procedure Install Satellite Forms on your development computer

1 Run the Satellite Forms **Setup** program.

- The Setup program normally runs automatically when you insert the CD into your CD-ROM drive. If it does not run automatically, browse to the **SF8_Install.exe** file on the CD and double-click it. If you downloaded the Satellite Forms installation file, browse to the folder into which you downloaded the file **SF8_Install.exe**, and double-click it to start the installation.

2 Follow the prompts to complete the installation.

- When the **Setup Type** dialog box appears, choose an installation type.

You can choose to install all program features or specify a custom installation by following the installation wizard prompts. You can select a different installation folder using the Custom option.

Installing the Satellite Forms engine on handheld devices

The Satellite Forms runtime engine is the program on your handheld device that runs the applications you create with MobileApp Designer. After you have installed Satellite Forms development tools, you also need to install the runtime engine(s) on your handheld devices. There are four engines:

- Pocket PC SDK
- Pocket PC RDK
- Palm OS SDK
- Palm OS RDK

Generally, you would use the SDK engine during development and use the RDK engine for final testing and when you distribute your application to users. The SDK (software development kit) runtime engine displays the Satellite Forms icon on the handheld device, while the RDK (redistribution kit) runtime engine is hidden on the handheld so that only your own application icon is displayed.

Note Starting with Satellite Forms 8, a new integrated runtime engine feature is now available. This combines the RDK runtime engine with your application icon into a single integrated runtime file, and you do not need to install the RDK runtime engine separately. For development purposes, you still need to install the SDK runtime, but for deployment you can use the integrated runtime engine.

Note The Trial version of Satellite Forms does not include the RDK runtime engines; the Trial version include the SDK runtime engines only.

Note If a previous version of a Satellite Forms Engine is installed on your handheld device, you must delete the old engine before installing the new one. For instructions, see Uninstall the Satellite Forms Engine from a Palm OS device on page 37 and/or Uninstall the Satellite Forms Engine from a Pocket PC device on page 37.

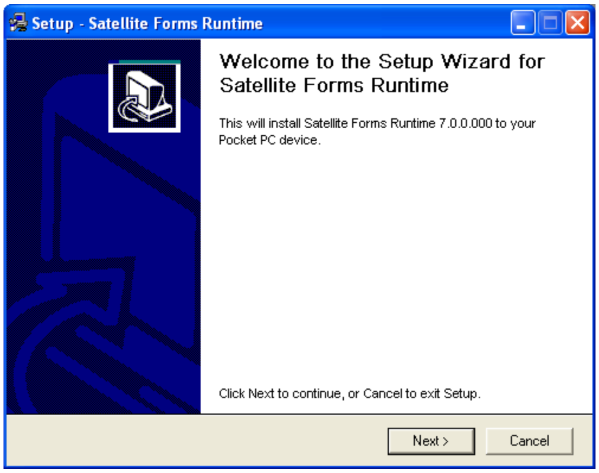
Installing the Satellite Forms Pocket PC SDK or RDK engine

Procedure Install the Pocket PC SDK or RDK engine

1 Click the **Start** button on the Windows Taskbar, point to **Programs** > **Satellite Forms 8.0** > **Runtime** > **Pocket PC**, and click **RDK Runtime Installer** or **SDK Runtime Installer**.

2 Follow the prompts to complete the installation.

The **SDK/RDK Runtime Installer** dialog box appears, similar to the one shown in the following figure:



Setup - Satellite Forms Runtime

Welcome to the Setup Wizard for Satellite Forms Runtime

This will install Satellite Forms Runtime 7.0.0.000 to your Pocket PC device.

Click Next to continue, or Cancel to exit Setup.

Installing the Satellite Forms Palm OS SDK or RDK engine

Use the SDK engine during development of your applications. When you distribute your application to your users, or for final testing, use the integrated RDK engine.

Procedure Install the Satellite Forms SDK or RDK engine on a Palm OS device

1 Start HotSync Manager if it is not already running.

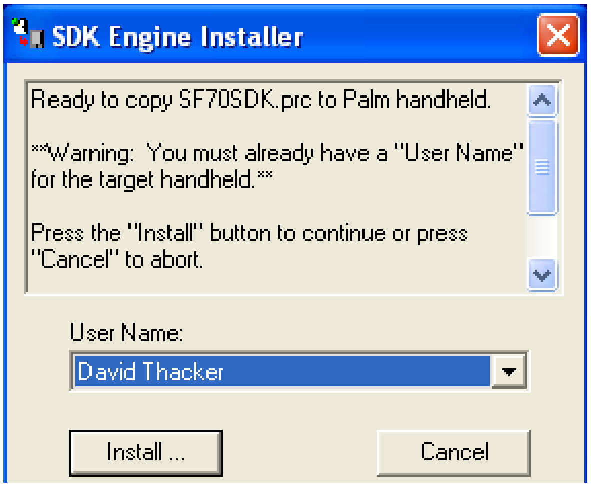
- Click the **Start** button on the Windows Taskbar, point to **Programs** > **Palm Desktop**, and then click **HotSync Manager**.

2 Click the Start button on the Windows Taskbar, point to **Programs** > **Satellite Forms 8.0** > **Runtime** > **Palm** and click the appropriate command for the program you want to install:

- **Install SDK Engine**
- **Install RDK Engine**

The **SDK/RDK Installer** dialog box appears, similar to the one shown in the following figure:

Figure 3.1 SDK Installer dialog box



Ready to copy SF70SDK.prc to Palm handheld.

Warning: You must already have a "User Name" for the target handheld.

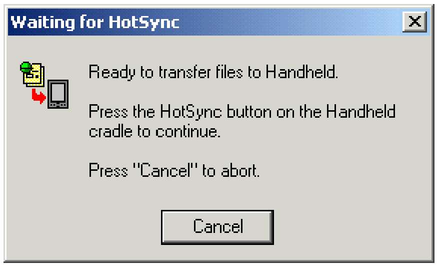
Press the "Install" button to continue or press "Cancel" to abort.

User Name: David Thacker

3 Select the **User Name** of the handheld device on which you want to install the SDK or RDK, and then click the **Install...** button.

The **Waiting for HotSync** dialog box appears, as shown in the following figure:

Figure 3.2 Waiting for HotSync dialog box



Ready to transfer files to Handheld.

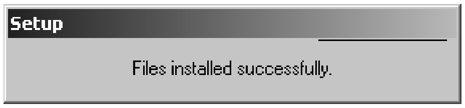
Press the HotSync button on the Handheld cradle to continue.

Press "Cancel" to abort.

4 Place the handheld in its cradle and press the **HotSync** button.

When the HotSync is complete, a confirmation message appears, as shown in the following figure. If this message does not appear or if an error message is displayed, repeat the Satellite Forms Engine install procedure.

Figure 3.3 Palm OS engine setup successful message box



Files installed successfully.

5 When the HotSync session is complete, remove the handheld device from the cradle and tap **Applications**.

The Satellite Forms icon appears on the handheld device's application picker screen. If the Satellite Forms icon does not appear, repeat the Satellite Forms Engine install procedure.

6 Tap the **Satellite Forms** icon to open the RDK/SDK program.

The **Satellite Forms** screen appears with an empty **Select Application to Run** list.

7 Tap the **Applications** button to return to the picker screen. Then return the handheld device to its cradle.

8 To confirm the installation, download and test a sample Satellite Forms application, as described in the following section.

Downloading and testing a Satellite Forms application

Procedure Download and test a Satellite Forms application

1 Open MobileApp Designer by double-clicking the **Satellite Forms 8.0** icon on your development computer's desktop.

- If the icon is not on your desktop, click the **Start** button on the Windows Taskbar, point to **Programs > Satellite Forms 8.0**, and click **MobileApp Designer**.

2 Select **File > Open** from the MobileApp Designer menu.

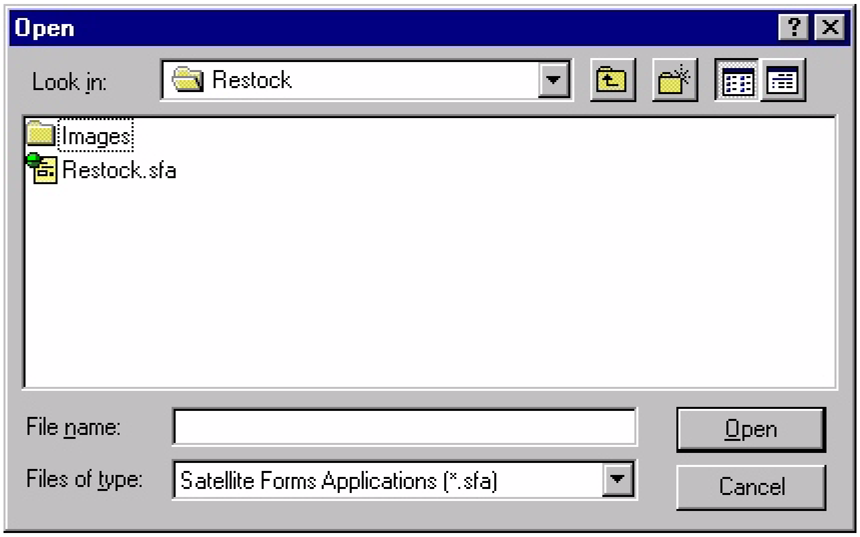
The **Open** dialog box appears.

3 Using the Open dialog box, browse to the **Restock** folder.

The path to the Restock folder is <*Satellite Form 8.0 install directory*>\Samples\Projects\Restock. If you installed Satellite Forms in the default location, the full path is:
C:\Satellite Forms 8.0\Samples\Projects\Restock\

The file Open dialog box appears as shown in the following figure:

Figure 3.4 Open dialog box



4 Click **Restock.sfa** and click the **Open** button to open the project.

5 Select **Handheld > Download App & Tables...** from the MobileApp Designer menu (or press the **F5** function key).

If the **Save table <table name> as dBase file** dialog box appears, click **Save** until the **Ready to synchronize with handheld** message box appears.

6 Press the **HotSync** button on the handheld cradle (this step is required for PalmOS devices only, and is not needed for Windows Mobile/Pocket PC devices).

7 When the HotSync or ActiveSync transfer is finished, remove the handheld device from the cradle and tap the **Applications** button (PalmOS) or **Start > Programs** (Pocket PC).

8 Open Satellite Forms by tapping the **Sat. Forms** icon

9 From the Select Application to Run list, tap **Store Restocking Demo**.

The **Restocking Visits** screen appears and in the Stores to Visit list, Joe's Food Mart is selected.

10 Tap the **Info** button.

The Customer Info screen appears.

11 Tap the **Notes** button.

The Customer Notes screen appears and displays Offer discount on Pretzel Products.

12 Tap **OK** to return to the Customer Info screen.

13 Tap **OK** to return to the Stores to Visit list.

14 Tap the **Orders** button.

The current order list includes Pretzels and a quantity of 500.

15 Tap the **Add** button.

The Select Category screen appears.

16 Tap the **Chips & Snacks** button.

The Chips & Snacks screen appears.

17 Select **Potato Chips**, enter **4** on the Quantity line using pen input or the handheld keyboard, and then tap the **OK** button.

The Orders screen now includes Potato Chips with a quantity of 4.

18 Tap the **Done** button.

19 Tap the **Menu** button and select **Options > Exit** to exit Satellite Forms.

Uninstalling Satellite Forms

The following sections provide instructions for uninstalling Satellite Forms MobileApp Designer from your desktop computer, deleting Satellite Forms applications from handheld devices, and uninstalling the Satellite Forms engine from handheld devices.

Uninstalling MobileApp Designer

You can uninstall MobileApp Designer at any time by using the Windows **Add/ Remove Program** option in the Windows Control Panel.

Procedure Uninstall MobileApp Designer

1 Click the **Start** button on the Windows Taskbar, point to **Settings**, click **Control Panel**, and double-click the **Add/Remove Programs** icon.

2 Select **Satellite Forms 8.0** from the programs listed, click the **Change/Remove** or **Add/Remove** button (depending on your version of Windows), and follow the prompts to uninstall the software.

Uninstalling the Satellite Forms Engine from Palm OS devices

Procedure Uninstall the Satellite Forms Engine from a Palm OS device

1 Tap the **Menu** icon on the handheld application picker screen.

2 Select **Delete**.

3 Select **Sat. Forms** from the Delete list.

4 Tap the **Delete** button.

5 Tap **Yes** to confirm that you want to delete the Satellite Forms Engine or tap **No** if you decide not to delete the engine.

Tap the **Done** button to return to the application picker screen.

Uninstalling the Satellite Forms Engine from Pocket PC devices

Procedure Uninstall the Satellite Forms Engine from a Pocket PC device

1 Tap the **Start | Settings** menu choice on the device.

2 Select the **System** tab and tap on **Remove Programs**.

3 Select **Thacker Satellite Forms Runtime** from the list.

4 Tap the **Remove** button.

5 Tap **Yes** to confirm that you want to delete the Satellite Forms Engine or tap **No** if you decide not to delete the engine.

Chapter 4
Quick Tour

Overview

This chapter is a step-by-step tutorial that shows you how to use MobileApp Designer to create a simple Satellite Forms application that contains one table and two forms. When the application is finished, you can download it to your handheld device and try it out. In this tutorial, you will not integrate the handheld application with a database application. For information on integrating handheld applications with a database application, see Chapter , Integrating with your Database, on page 195.

This Quick Tour leads you through the steps to create a Windows Mobile/Pocket PC application, but the same steps are used to create a PalmOS application: the skills you learn can be applied to either mobile platform.

The Quick Tour includes eight major steps:

1. Create a new project and select a handheld device target for it.
2. Create a table in which to store information.
3. Create the Main form and add controls to it.
4. Create the Notes form and add controls to it.
5. Assign actions to the Button controls.
6. Set the application's properties.
7. Download the application to a handheld device and test it.
8. Upload the modified table to your development computer and verify changes.

Step 1. Opening a new project

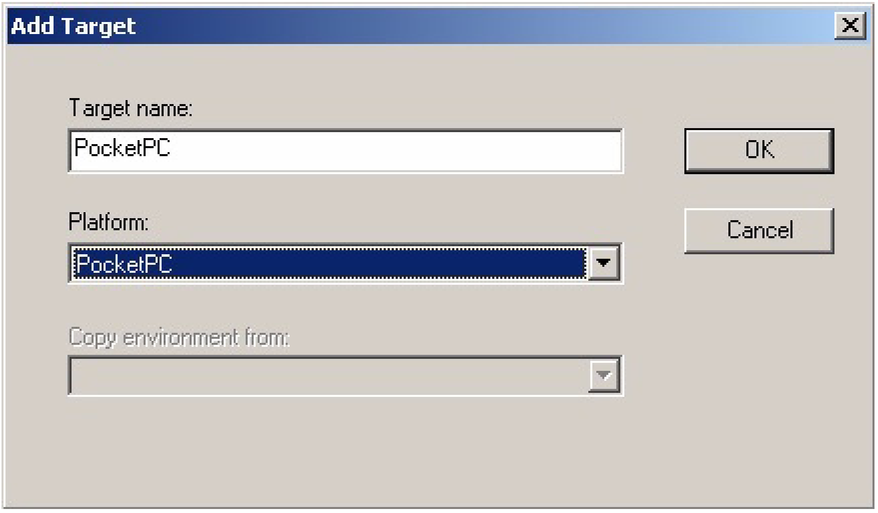
The first step is to open a new project. A project can contain one or more application targets, each of which can have a distinct name, if desired. An application target is a specific set of configurations for a handheld device and OS. For example, you can create a project with application targets for PalmOS and Pocket PC devices. You can then build the project for one application target at a time or for both application targets simultaneously.

1 Run MobileApp Designer. Click the **Start** button on the Windows Taskbar, point to **Programs > Satellite Forms 8.0**, and click **MobileApp Designer**.

2 Create a new Project. Select **File > New Project...** from the MobileApp Designer menu or click the **New Project** button on the **General** toolbar.

The **Add Target** dialog box appears as shown in the following figure:

Figure 4.1 Add Target dialog box



Add Target

Target name: PocketPC

Platform: PocketPC

Copy environment from:

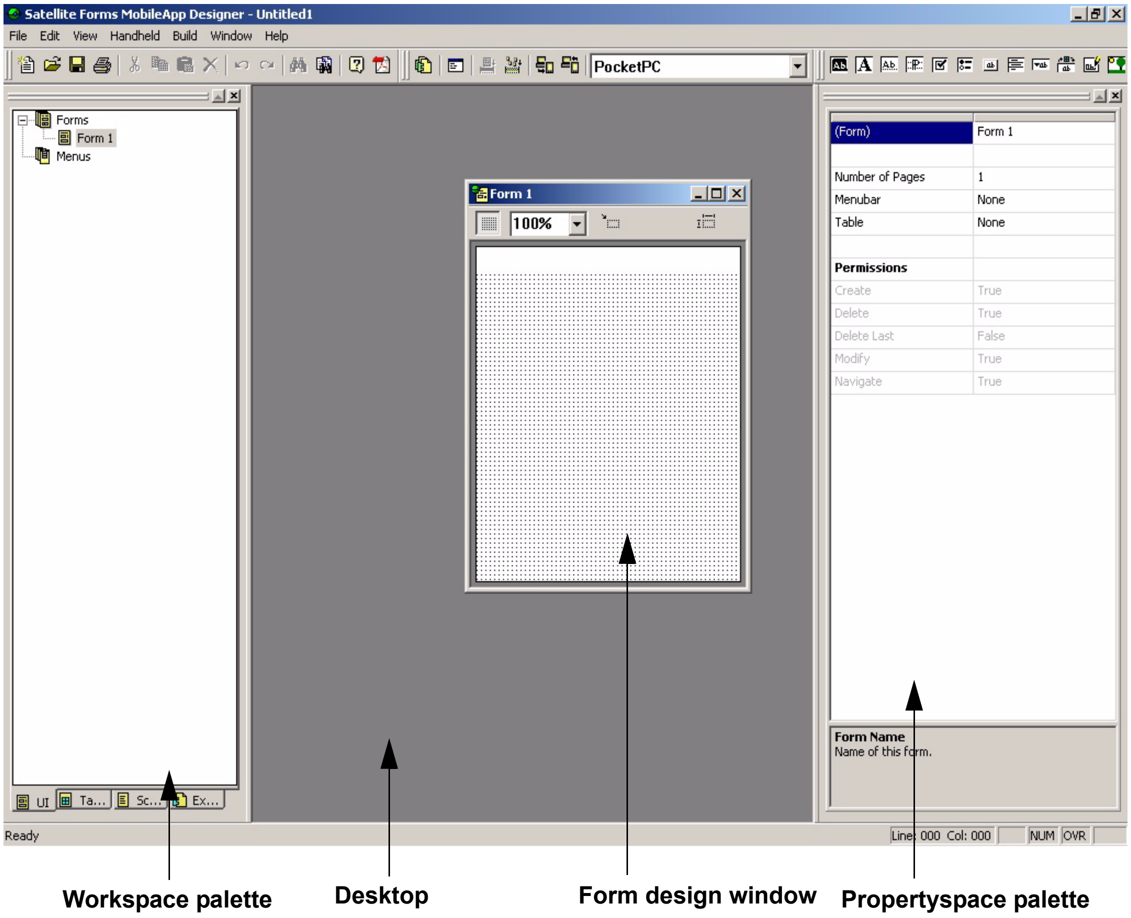
3 In the **Add Target** dialog box, set the following options and then click the **OK** button:

- **Target name:** A name unique to the project that specifies the target, for example, Palm or PocketPC.
- **Platform:** Select the target for the new project. For this tutorial, select **PocketPC**. The Target name is updated to the default for that target when you select it, which you can then edit if desired.

Tip Select **Build > Targets...** from the MobileApp Designer menu to add another target platform to a project. You can then use the **Build > Batch Build** menu option to build applications for all targets from a project in a single step.

MobileApp Designer then opens a default project with a blank form labeled **Form 1**, as shown in the following figure:

Figure 4.2 MobileApp Designer with new project open



(Form)	Form 1
Number of Pages	1
Menubar	None
Table	None
Permissions	
Create	True
Delete	True
Delete Last	False
Modify	True
Navigate	True

Form Name
Name of this form.

Step 2. Creating the CtvCustomers table

Next, create a table for the application's data. Projects can use more than one table, but this tutorial uses just one simple table.

1 Select **Edit > Insert Table** from the MobileApp Designer menu.

The **Table** dialog box appears, similar to the one shown in the following figure:

Figure 4.3 Table dialog box

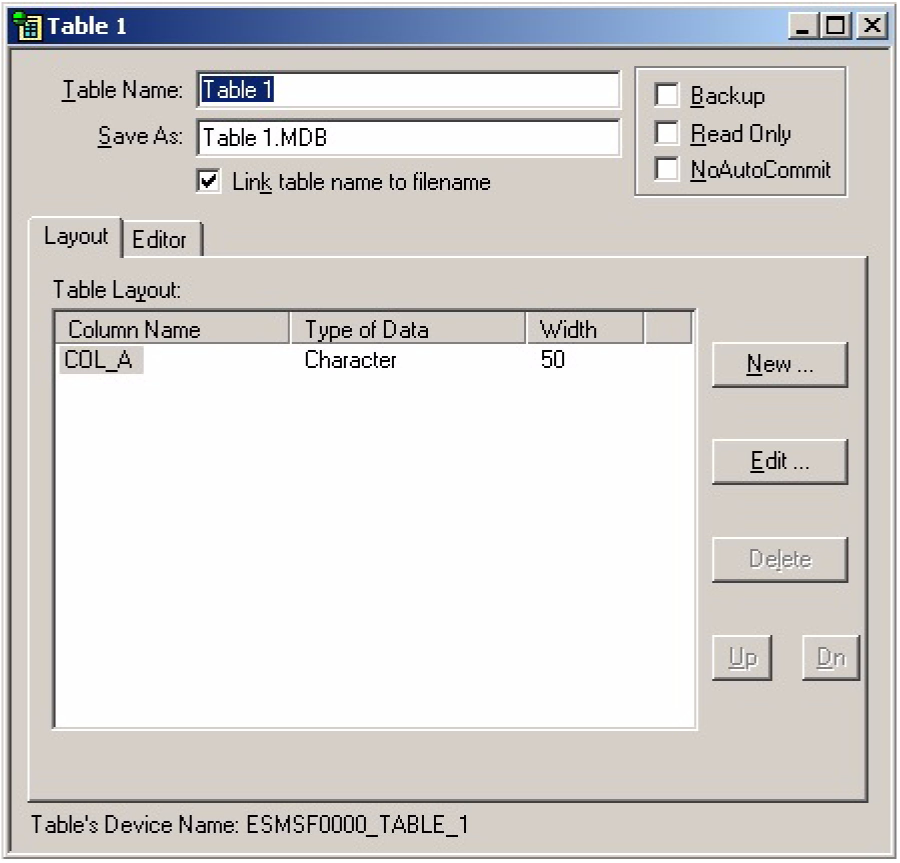


Table Name: Table 1

Save As: Table 1.MDB

☑ Link table name to filename

□ Backup

□ Read Only

□ NoAutoCommit

Table Layout:

Column Name	Type of Data	Width
COL_A	Character	50

Table's Device Name: ESMSF0000_TABLE_1

The **Table** dialog box contains the following tabs:

- **Layout:** Add, remove, and edit columns or move columns up or down in the table.
- **Editor:** Add data to a table to perform simple tests in your application.

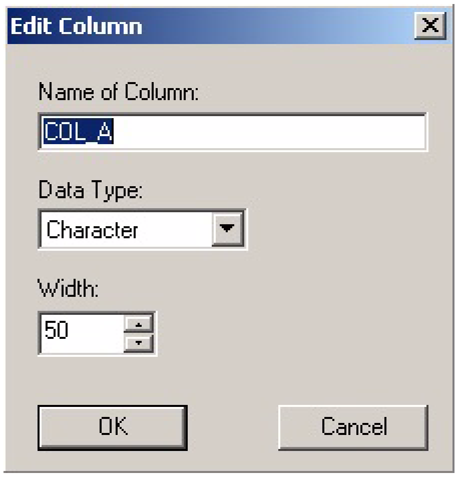
2 Name the table by typing `CtvCustomers` in the **Table Name** edit box.

Ctv stands for Customers to Visit, which is the name of this sample application. The name `CtvCustomers` ensures that the table name is unique. If two tables on a handheld device have the same name, even if they are used by different applications, they will overwrite each other when information is synchronized. Satellite Forms recommends that you begin all table names with a unique prefix. An abbreviation of the application name generally makes a good prefix.

When you create a new table, MobileApp Designer inserts a default column, labeled **COL_A**. Begin by editing that column.

3 Click the **Edit** button to open the **Edit Column** dialog box, as shown in the following figure:

Figure 4.4 Edit Column dialog box



Name of Column: COL_A

Data Type: Character

Width: 50

4 In the **Edit Column** dialog box, edit the fields as follows and then click the **OK** button:

- **Name of Column:** Type `ID`.
- **Data Type:** Select **Numeric**.
- **Num. Decimals:** Use the default, **None**.
- **Width:** Use the default, **4**.

The **ID** column appears on the **Layout** tab.

5 Next, click the **New** button to open the **Create New Column** dialog box, with which you can begin adding more columns.

Tip The **Create New Column** dialog box looks and functions like the **Edit Column** box shown above.

6 Create the following columns:

- Name of Column: **NAME**; Data Type: **Character**; Width: **20**.
- Name of Column: **CITY**; Data Type: **Character**; Width: **30**.
- Name of Column: **PHONE**; Data Type: **Character**; Width: **12**.
- Name of Column: **NOTES**; Data Type: **Character**; Width: **512**.

When you have finished adding these columns, your table layout should look similar to the one shown in the following figure:

Figure 4.5 Table dialog box, Layout tab

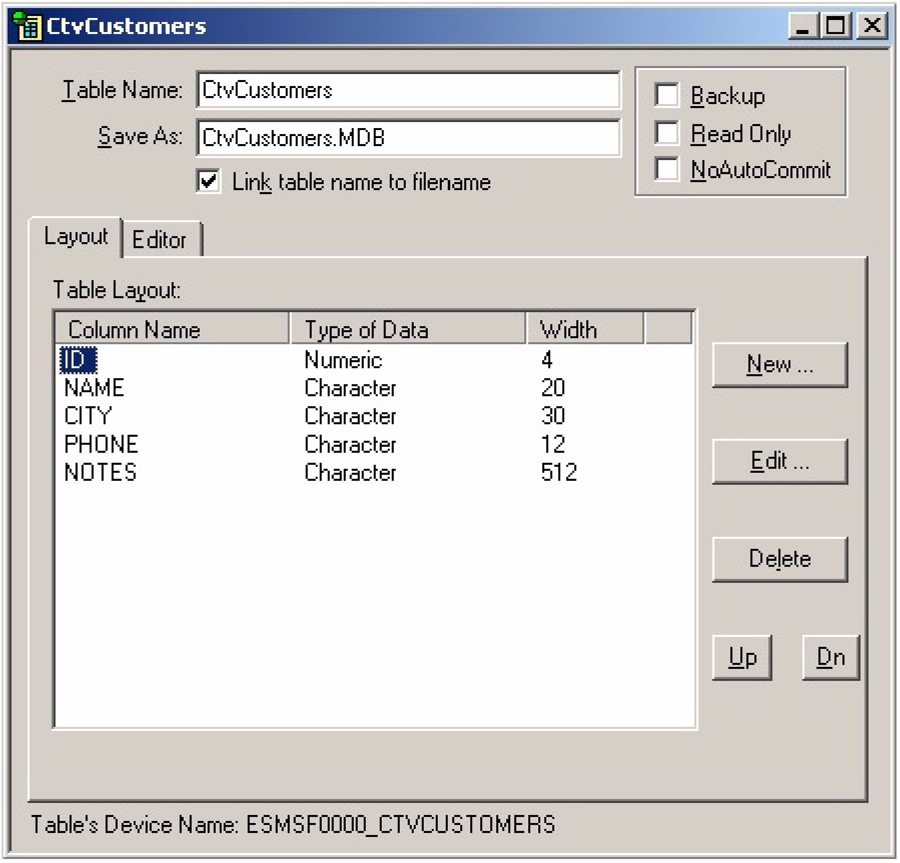


Table Name: CtvCustomers

Save As: CtvCustomers.MDB

☑ Link table name to filename

☐ Backup
☐ Read Only
☐ NoAutoCommit

Table Layout:

Column Name	Type of Data	Width
ID	Numeric	4
NAME	Character	20
CITY	Character	30
PHONE	Character	12
NOTES	Character	512

Table's Device Name: ESMSF0000_CTVCUSTOMERS

Next, add some sample information to the table. This data is used when you test the application later.

7 To add information, click the **Editor** tab as shown in the following figure:

Figure 4.6 Table dialog box, Editor tab

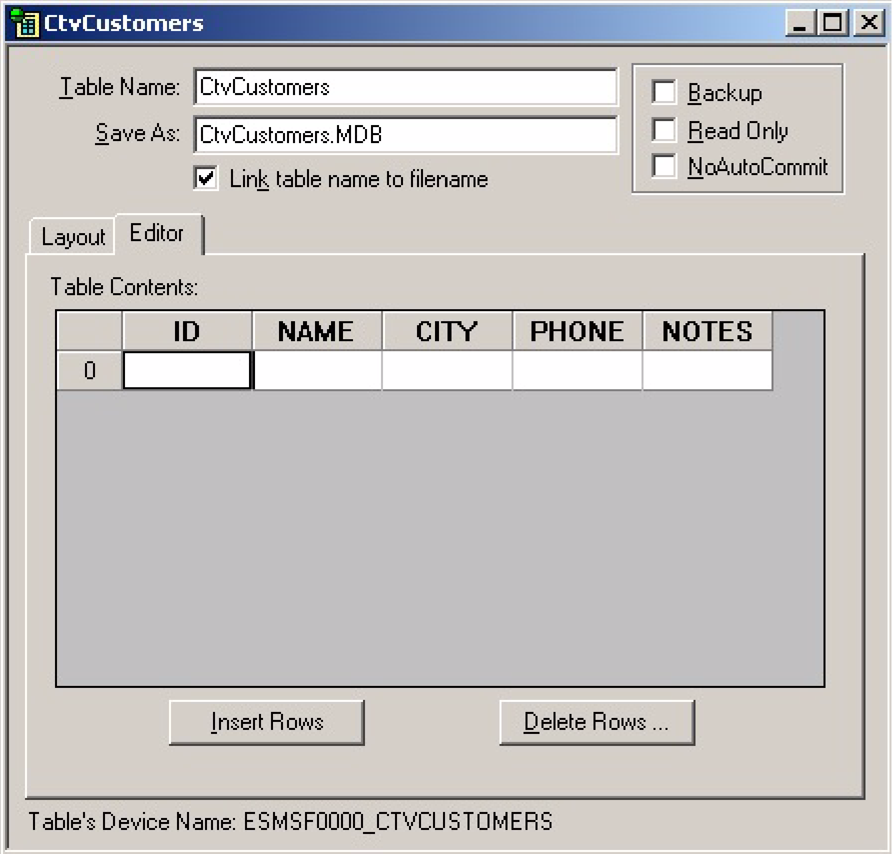


Table Name: CtvCustomers

Save As: CtvCustomers.MDB

☑ Link table name to filename

☐ Backup
☐ Read Only
☐ NoAutoCommit

Table Contents:

	ID	NAME	CITY	PHONE	NOTES
0					

Table's Device Name: ESMSF0000_CTVCUSTOMERS

8 Next, click in the **ID** field, type a `1` in the cell, and then press the **Tab** key on your keyboard to advance to the next field.

Tabbing to the last field of the last row or clicking the **down-arrow** key in the last row adds a new row.

9 Enter the following data into the CtvCustomers table:

Table 4.1 CtvCustomers table, sample data

ID	NAME	CITY	PHONE	NOTES
1	Mohammed’s	Rochester	315-555-4393	Check inventory
2	Nicole’s Place	Cobleskill	518-555-1234	Show catalog
3	The Outrigger	Winterset	518-555-4563	Pick up returns
4	The Fortune	Buffalo	315-555-5896	Pick up check
5	Joe’s Diner	Albany	587-555-5487	Talk up specials

10 Adjust the column widths to show all of the information.

To adjust the column width, click and hold the divider between the **NAME** column and the **CITY** column and drag it to the right until the full names are displayed. Then do the same with the divider at the right-hand side of the **CITY**, **PHONE**, and **NOTES** columns. You can also shrink the **ID** column by clicking the divider between **ID** and **NAME** and dragging it to the left.

The following figure shows an example of a table after data entry and column width adjustment:

Figure 4.7 CtvCustomers table with data entered

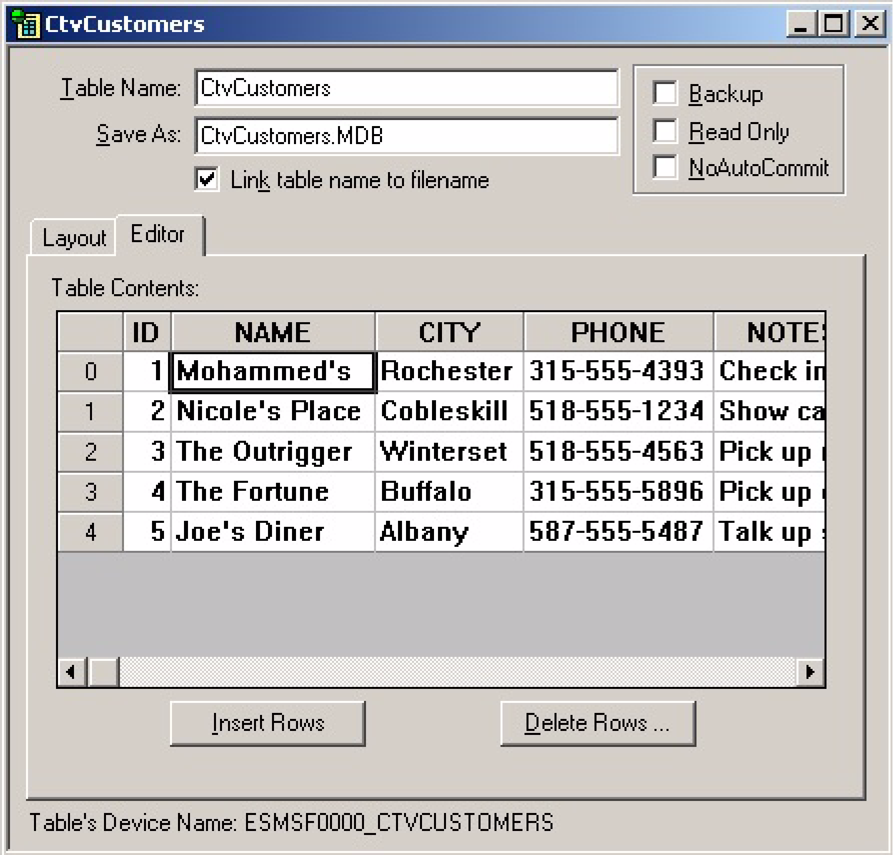


Table Name: CtvCustomers

Save As: CtvCustomers.MDB

☑ Link table name to filename

□ Backup

□ Read Only

□ NoAutoCommit

Table Contents:

	ID	NAME	CITY	PHONE	NOTE
0	1	Mohammed's	Rochester	315-555-4393	Check in
1	2	Nicole's Place	Cobleskill	518-555-1234	Show ca
2	3	The Outrigger	Winterset	518-555-4563	Pick up
3	4	The Fortune	Buffalo	315-555-5896	Pick up
4	5	Joe's Diner	Albany	587-555-5487	Talk up

Table's Device Name: ESMSF0000_CTVCUSTOMERS

11 When you are finished adjusting the column sizes, click the close button (**X**) in the upper-right corner of the dialog box title bar to close the table editor.

Tip Save frequently while you are working in MobileApp Designer. Satellite Forms also recommends creating a unique directory for each project.

12 Save the application by completing the following steps:

a Open the **Save As** dialog box using one of the following methods:

- Select **File > Save** from the menu bar.
- Click the **Save** button on the **General** toolbar.
- Press **Ctrl + S** on the keyboard.

b Right-click in the **Save As** dialog box and select **New > Folder**.

c Name the new folder `Customers to Visit`.

d Double-click the **Customers to Visit** folder to open it.

e Type `Customers` in the File Name edit box and then click the **Save** button.

The dialog box closes and **Customers.sfa** appears in the MobileApp Designer title bar.

Step 3. Creating the Main form

In this step, you use the default form MobileApp Designer created to build the main form that users work with when using the Customers to Visit application. First, set the form properties, then add a Title control, three Text controls, three Edit controls, and a Button control.

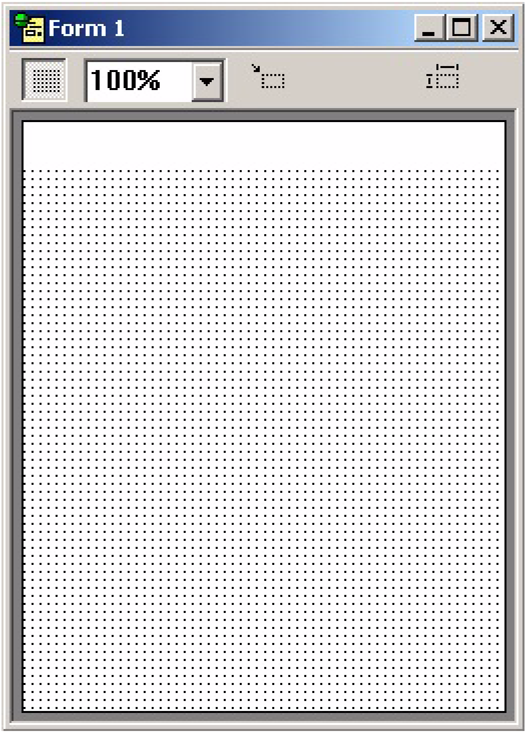
Tip You can create a new form by selecting **Edit > Insert Form** from the MobileApp Designer menu.

Setting form properties

1 If Form 1 does not appear on the desktop, click the **UI** tab on the **Workspace** palette and double-click the **Form 1** icon.

Form 1 is blank, as shown in the following figure:

Figure 4.8 Form design window



Tip The form design window will look slightly different for a PalmOS application target, because the forms are normally square for the PalmOS platform.

2 The next step is to set the form properties. Type the following information in the value (right-hand) column of the **Propertyspace** palette:

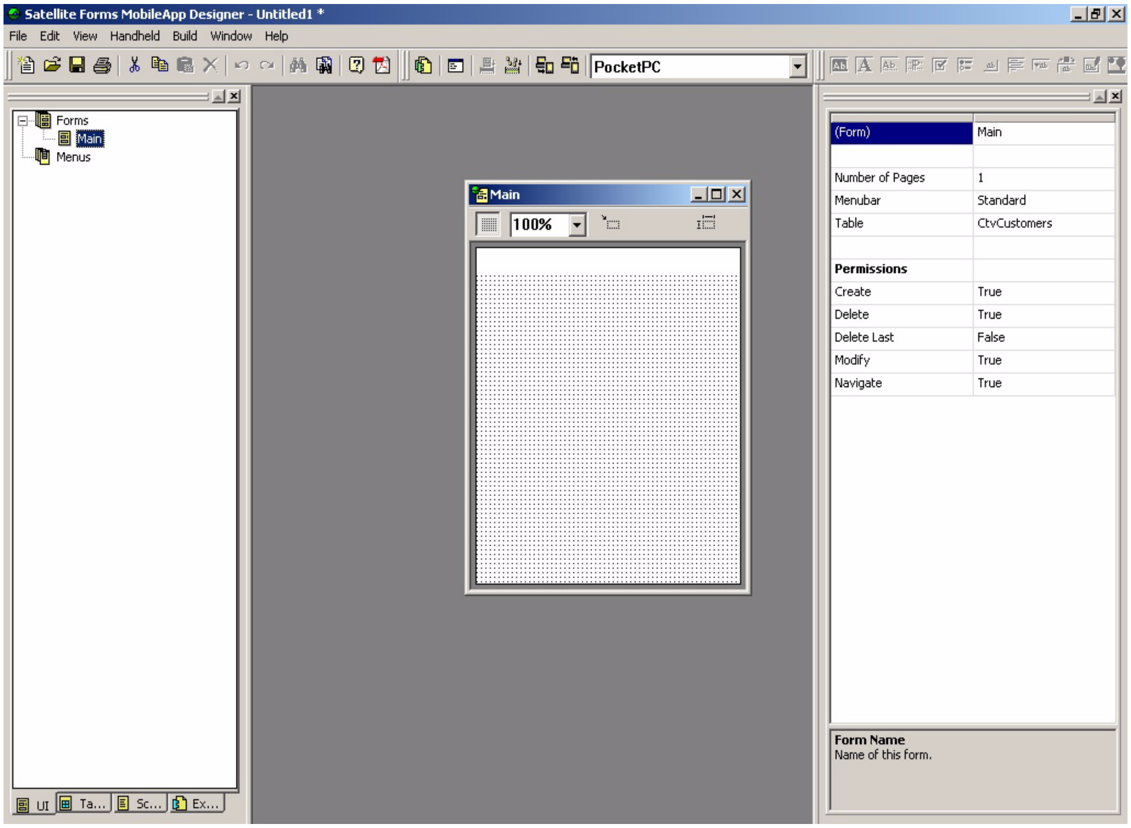
- **(Form):** Double-click and type `Main`.
- **Number of Pages:** Use the default: **1**.
- **Menubar:** Click the value field, which currently reads **None**, click the **down-arrow button**, and then select **Standard**.

- **Table:** Click the value field, which currently reads **None**, click the **down-arrow button**, click the **lower spin button** in the combo box, and then click **CtvCustomers**.

  This links the form to the CtvCustomers table, which means that the form displays data from that source. When you select the linked table, the **User Permissions** section becomes active. Do not change the default settings in the **User Permissions** section.

The MobileApp Designer window should now look like the example shown in the following figure. Note that the form is selected on the desktop and that the **Propertyspace** and **Workspace** palettes are updated.

Figure 4.9 Main form



(Form)	Main
Number of Pages	1
Menubar	Standard
Table	CtvCustomers
Permissions	
Create	True
Delete	True
Delete Last	False
Modify	True
Navigate	True

Form Name
Name of this form.

Inserting the title control

Next, add a title to the form. The Title control sets the text that appears on the form title bar when users are working with the application.

3 Click the **Title Control** button on the **Control Palette** toolbar.

A Title control appears at the top of the form at a fixed position. The Title control is the selected object and the Propertyspace palette now shows the Title control properties.

4 In the **Propertyspace** palette, edit the following fields to set the Title control properties and then click the Title control:

- **(Title):** Double-click and type `Customers_to_Visit`.

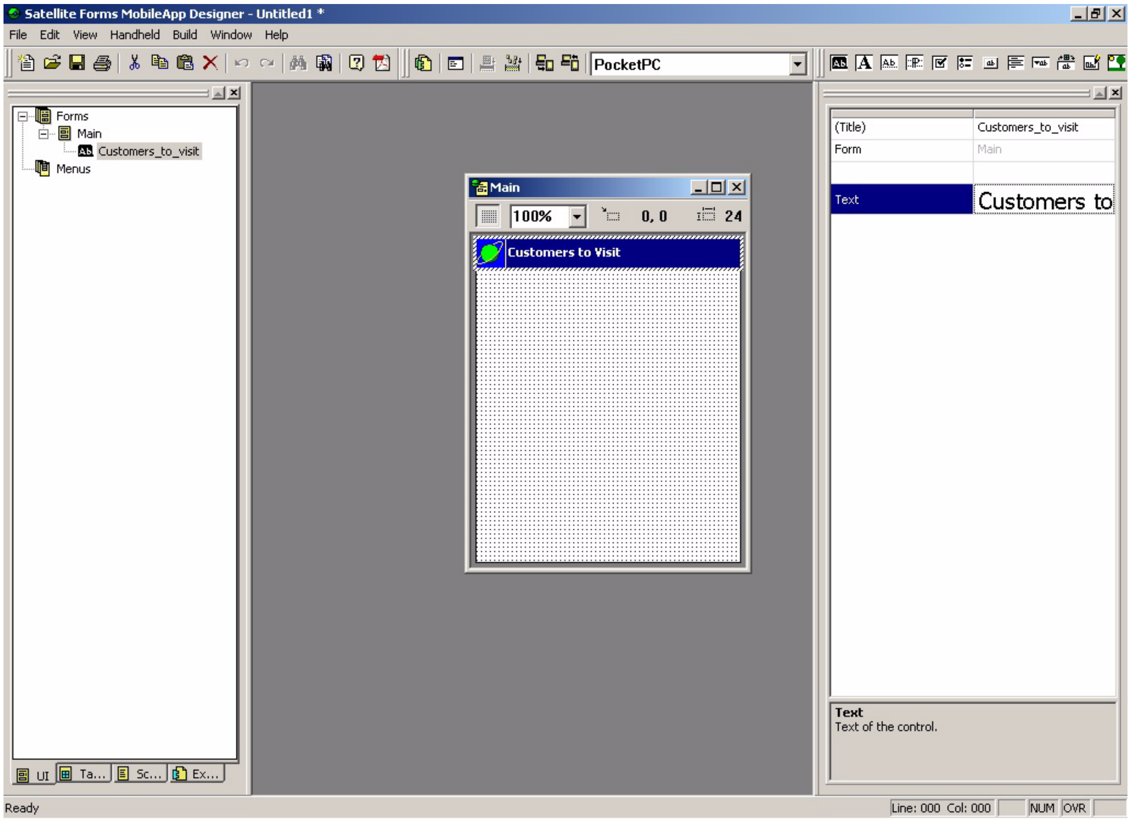
Control names cannot contain spaces. The name appears next to the control's icon on the UI tab of the Workspace palette.

- **Text:** Type `Customers to Visit`.

This text appears on the form.

The MobileApp Designer window should now look like the example shown in the following figure. Note that the Title control is selected on the form and that the Propertyspace and Workspace palettes are updated.

Figure 4.10 Main form with Title control



(Title)	Customers_to_visit
Form	Main
Text	Customers to

Text
Text of the control.

Inserting Text controls and Edit controls

Next, add fields in which the information in the NAME, CITY, and PHONE columns of the table can be displayed and edited. To do this, add three Text controls and three Edit controls. Text controls display static text on a form. They are useful for providing information or labels for other controls. On this form, the Text controls label the Edit controls. Edit controls display and allow the editing of data in a column in the form's linked table.

5 Click the **Text Control** button A on the **Control Palette** toolbar.

A Text control appears below the Title control. The Text control is the selected object and the **Propertyspace** palette shows the Text control properties.

6 In the **Propertyspace** palette, edit the following fields to set the Text control properties and then click the Text control.

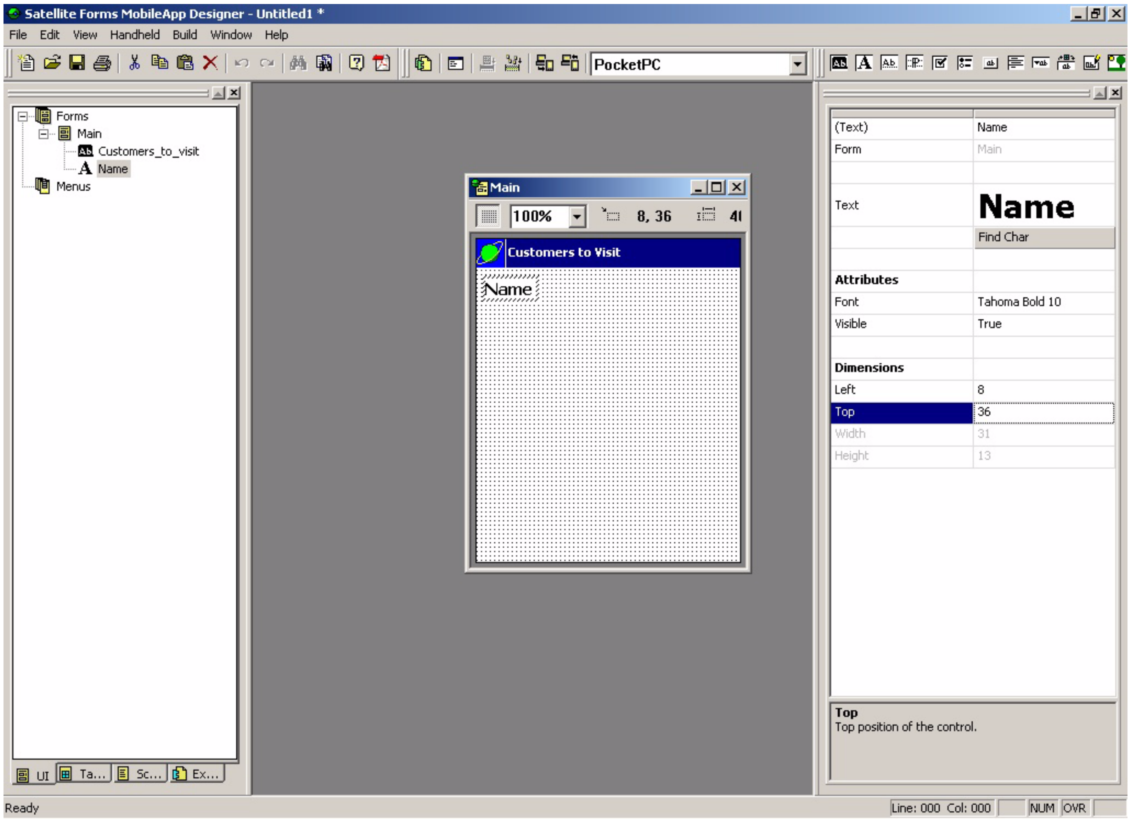
- **(Text):** Double-click and type `Name`.
- **Text:** Double-click and type `Name`.

- **Font:** Click and select **Tahoma Bold 10** from the list.
- **Visible:** Use the default value, **True**.
- **Left:** Double-click and type `8`.
- **Top:** Double-click and type `36`.
- *For a PalmOS target, choose the Bold 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*

Tip You can also position controls by dragging them on the form. When a control is selected, the current x-y coordinates are listed at the top of the Form design window.

The MobileApp Designer window should now look like the example shown in the following figure. Note that the Text control is selected on the form and that the Propertyspace and Workspace palettes are updated.

Figure 4.11 Main form with Text control



(Text)	Name
Form	Main
Text	Name Find Char
Attributes	
Font	Tahoma Bold 10
Visible	True
Dimensions	
Left	8
Top	36
Width	31
Height	13

Top
Top position of the control.

Ready Line: 000 Col: 000 NUM OVR

Next, add the first Edit control.

7 Click the **Edit Control** button Ab on the **Control Palette** toolbar.

An Edit control appears below the Title control. The Edit control is the selected object and the **Propertyspace** palette shows the Edit control properties.

All new controls appear at the same location on the form. When you add new control, existing controls at that location are hidden. To show the existing control, set a different location for the new control using the Left and Top properties or simply click and drag the new control to another location on the form.

8 In the **Propertyspace** palette, edit the following fields to set the Edit control properties and then click the Edit control.

- **(Edit Text):** Double-click and type `Customer_Name`.
- **Column:** Click and select **NAME** from the list.

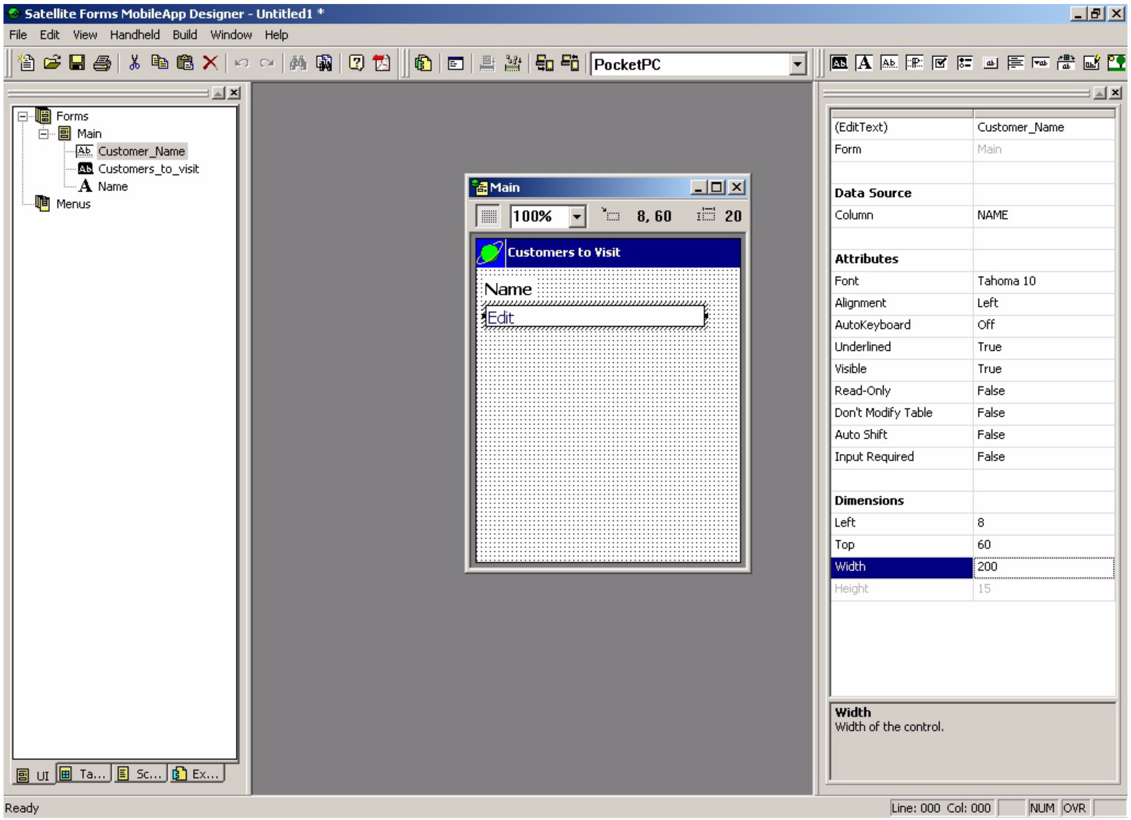
  This binds the Customer_Name Edit control to the NAME column in the CtvCustomers table.

- **Attributes:** Set the Font to **Tahoma 10** and use the default values for all other properties in the Attributes section.
- **Left:** Double-click and type `8`.
- **Top:** Double-click and type `60`.
- **Width:** Double-click and type `200`.
- *For a PalmOS target, choose the Normal 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*

Tip You can also resize a control by simply clicking and dragging the grapples on the left or right edge of the control. When you select a control, the current Width and Height properties are listed at the top of the Form design window.

The MobileApp Designer window should now look like the example shown in the following figure. Note that the Edit control is selected on the form and that the Propertyspace and Workspace palettes are updated.

Figure 4.12 Main form with Edit control



(EditText)	Customer_Name
Form	Main
Data Source	
Column	NAME
Attributes	
Font	Tahoma 10
Alignment	Left
AutoKeyboard	Off
Underlined	True
Visible	True
Read-Only	False
Don't Modify Table	False
Auto Shift	False
Input Required	False
Dimensions	
Left	8
Top	60
Width	200
Height	15

Width
Width of the control.

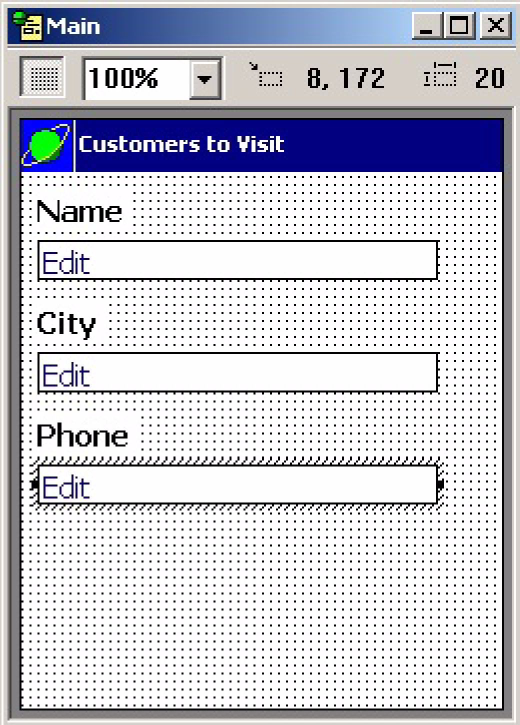
Next add Text controls and Edit controls for the CITY and PHONE fields.

9. To add the City text field, click the **Text Control** button A on the **Control Palette** toolbar and then edit the following fields in the **Propertyspace** palette:
   - **(Text):** Double-click and type `City_Name`.
   - **Text:** Double-click and type `City`.
   - **Font:** Click and select **Tahoma Bold 10** from the list.
   - **Visible:** Use the default value, **True**.
   - **Left:** Double-click and type `8`.
   - **Top:** Double-click and type `92`.
   - *For a PalmOS target, choose the Bold 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*
10. To add the City edit field, click the **Edit Control** button Ab on the **Control Palette** toolbar and then edit the following fields in the **Propertyspace** palette:
    - **(Edit Text):** Double-click and type `City`.
    - **Column:** Click and select **CITY** from the list.
    - **Attributes:** Set the Font to **Tahoma 10** and use the default values for all other properties in the Attributes section.
    - **Left:** Double-click and type `8`.
    - **Top:** Double-click and type `116`.
    - **Width:** Double-click and type `200`.
    - *For a PalmOS target, choose the Normal 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*
11. To add the Phone text field, click the **Text Control** button A on the **Control Palette** toolbar and then edit the following fields in the **Propertyspace** palette:
    - **(Text):** Double-click and type `Phone_Number`.
    - **Text:** Double-click and type `Phone`.
    - **Font:** Click and select **Tahoma Bold 10** from the list.
    - **Visible:** Use the default value, **True**.
    - **Left:** Double-click and type `8`.
    - **Top:** Double-click and type `148`.
    - *For a PalmOS target, choose the Bold 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*
12. To add the Phone edit field, click the **Edit Control** button Ab on the **Control Palette** toolbar and then edit the following fields in the **Propertyspace** palette:
    - **(Edit Text):** Double-click and type `Phone`.
    - **Column:** Click and select **PHONE** from the list. Use the spin buttons to scroll through the list of columns, if necessary.

- **Attributes:** Set the Font to **Tahoma 10** and use the default values for all other properties in the Attributes section.
- **Left:** Double-click and type `8`.
- **Top:** Double-click and type `172`.
- **Width:** Double-click and type `200`.
- *For a PalmOS target, choose the Normal 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*

When you have finished setting up these controls, the Main form should look like the example shown in the following figure:

Figure 4.13 Main form with Text and Edit controls



Inserting a Button control

Next, add a button labeled **Notes**. Information from the NOTES column of the CtvCustomers table is displayed on a separate form which users access by tapping the **Notes** button.

13 Click the **Button Control** button ab on the **Control Palette** toolbar and drag the control to the bottom of the form.

14 Edit the following fields in the **Propertyspace** palette:

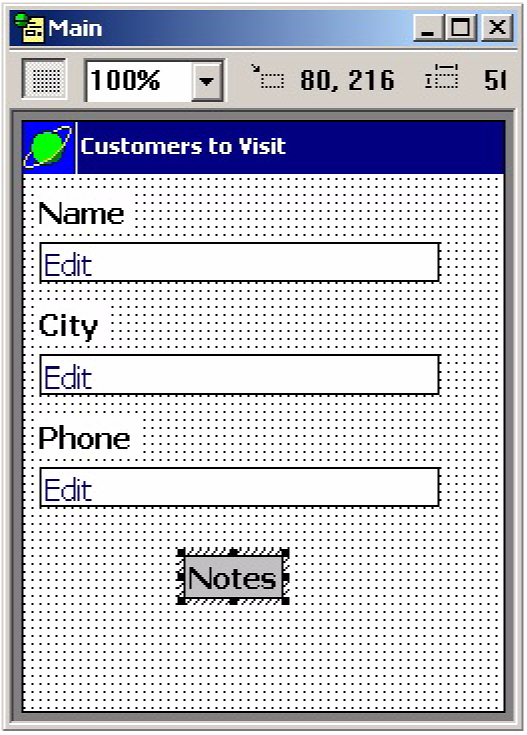
- **(Button):** Double-click and type `Notes`.
- **Text:** Double-click and type `Notes`.

  This text is the button label. The button automatically sets its size based on the label, but you can use the **Width** and **Height** properties to set the button size manually.

- **Attributes:** Set the Font to **Tahoma Bold 10** and use the default values for all other properties in the **Attributes** section.
- **Left:** Double-click and type `80`.
- **Top:** Double-click and type `216`.
- *For a PalmOS target, choose the Bold 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*

The Main form is now complete and should look like the example shown in the following figure:

Figure 4.14 Main form with Notes button



15 Close the Main form and continue with the following section.

Step 4. Creating the Notes form

Next, add a new form for data from the NOTES column of the CtvCustomers table. This form is for jotting down ideas or viewing important customer information.

Creating and configuring this form is similar to setting up the Main form. The process involves adding and configuring one Title control, one Paragraph control, and one Button control.

Creating the new form

First, add a new form to the application.

1 Select **Edit > Insert Form** from the MobileApp Designer menu.

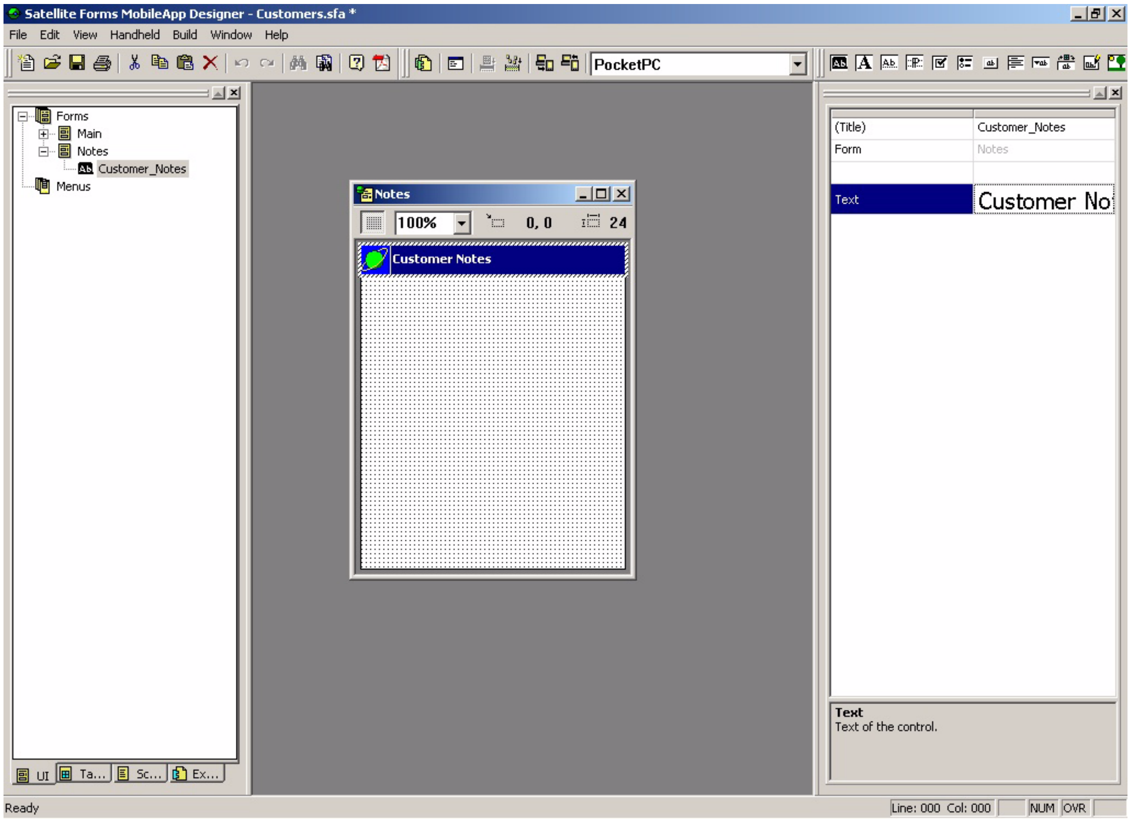
A new form labeled Form 2 appears on the desktop.

2 In the **Propertyspace** palette, set the following properties:

- **(Form):** Double-click and type `Notes`.
- **Number of Pages:** Use the default, **1**.
- **Menubar:** Click the value field, which currently reads **None**, click the **down-arrow button**, and then select **Standard**.
- **Table:** Click the value field, which currently reads **None**, click the down-arrow button, click the lower spin button in the drop-list, and then click **CtvCustomers**.
- **Permissions:** Set the **Modify** property to **True** so that the notes can be modified on the handheld, but set all other properties in the Permissions section to **False**.

The MobileApp Designer window should now look like the example shown in the following figure:

Figure 4.15 Notes form



(Title)	Customer_Notes
Form	Notes
Text	Customer No

Text
Text of the control.

Adding the Title control

Now add the Title control to the Notes form.

3 Click the **Title Control** button on the **Control Palette** toolbar.

A Title control appears at the top of the form at a fixed position. The Title control is the selected object and the Propertyspace palette now shows the Title control properties.

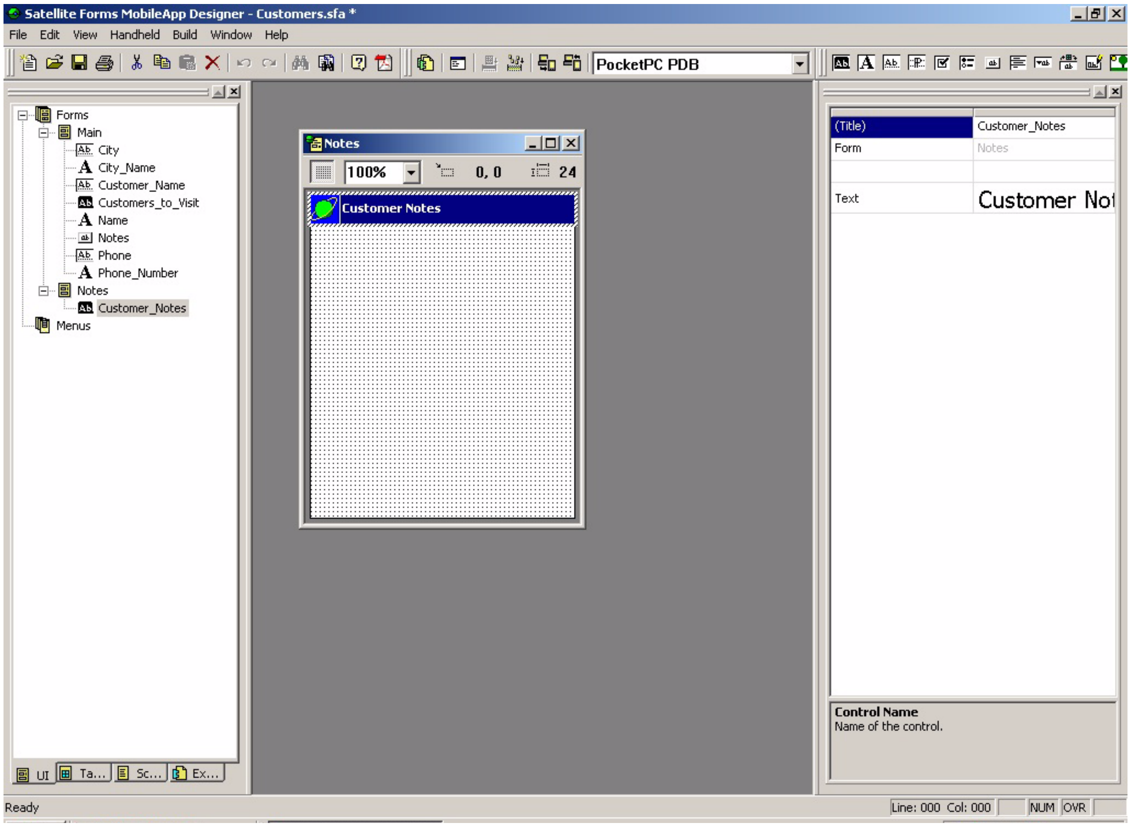
4 In the **Propertyspace** palette, edit the following fields to set the Title control properties and then click the Title control:

- **(Title):** Double-click and type `Customer_Notes`.

- **Text:** Type `Customer Notes`.

The MobileApp Designer window should now look like the example shown in the following figure:

Figure 4.16 Notes form with Title control



(Title)	Customer_Notes
Form	Notes
Text	Customer Not

Control Name
Name of the control.

Inserting the Paragraph control

Next, add a Paragraph control and link it to the NOTES column of the CtvCustomers table so that users can enter or view notes. Because the NOTES column can contain the equivalent of many lines of information, use a Paragraph control. A Paragraph control is similar to an Edit control, except that it can display and edit multiple lines of information.

5 Click the **Paragraph Control** button on the **Control Palette** toolbar.

A Paragraph control appears below the Title control. The Paragraph control is the selected object and the Propertyspace palette shows the Paragraph control properties.

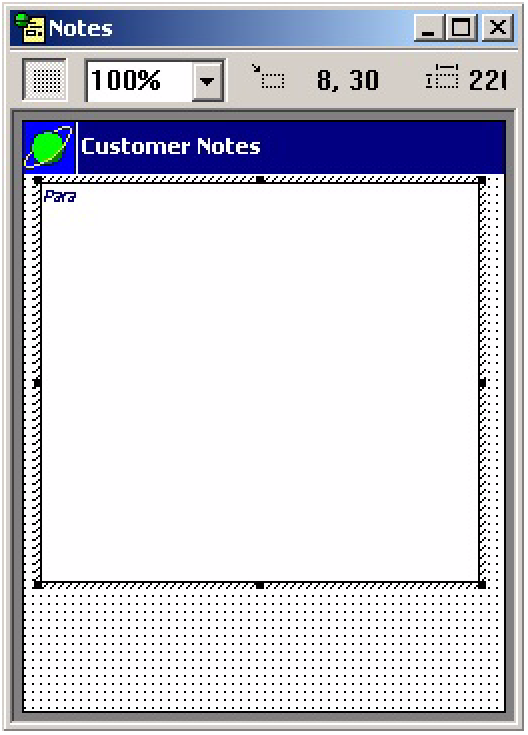
6 Edit the following fields in the **Propertyspace** palette and then click the Paragraph control:

- **(Paragraph):** Double-click and type `Notes_Paragraph`.
- **Column:** Click and select **NOTES** from the list. Use the spin buttons to scroll through the list of columns, if necessary.
- **Attributes:** Set the Font to **Tahoma 10** and use the default values for all other properties in the Attributes section.
- **Left:** Double-click and type `8`.

- **Top:** Double-click and type `30`.
- **Width:** Double-click and type `220`.
- **Height:** Double-click and type `200`.
- *For a PalmOS target, choose the Normal 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*

The Notes form should now look like the example shown in the following figure:

Figure 4.17 Notes form with Paragraph control



Inserting the Button control

Next add a Button control so users can close the form and return to the Main form when they finish entering or viewing data in the Notes field.

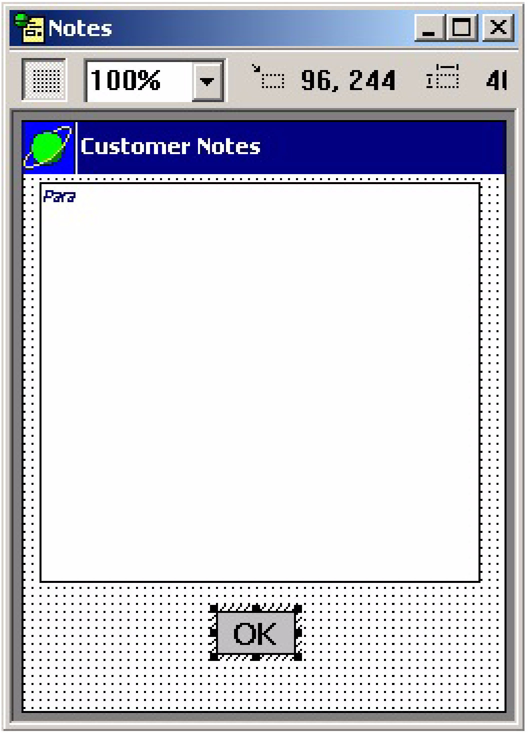
7 Click the **Button Control** button ab| on the **Control Palette** toolbar and then drag the control to the bottom of the form.

8 Edit the following fields in the **Propertyspace** palette:

- **(Button):** Double-click and type `OK`.
- **Text:** Double-click and type `OK`.
- **Attributes:** Set the Font to **Tahoma Bold 10** and use the default values for all other properties in the Attributes section.
- **Left:** Double-click and type `96`.
- **Top:** Double-click and type `244`.
- **Width:** Double-click and type `40`.
- *For a PalmOS target, choose the Bold 9 font, and set equivalent Left and Top values so that the form layout matches the sample screenshot.*

The Notes form is now complete and should look like the example shown in the following figure:

Figure 4.18 Notes form with OK button



9 Leave the Notes form open and continue with the following section.

Step 5. Assigning actions to the buttons

Next, assign an action to the two buttons in the application. Start with the Notes form because it is already open.

1 On the Notes form, click the **OK** button to select it.

The Button control properties appear in the Propertyspace palette.

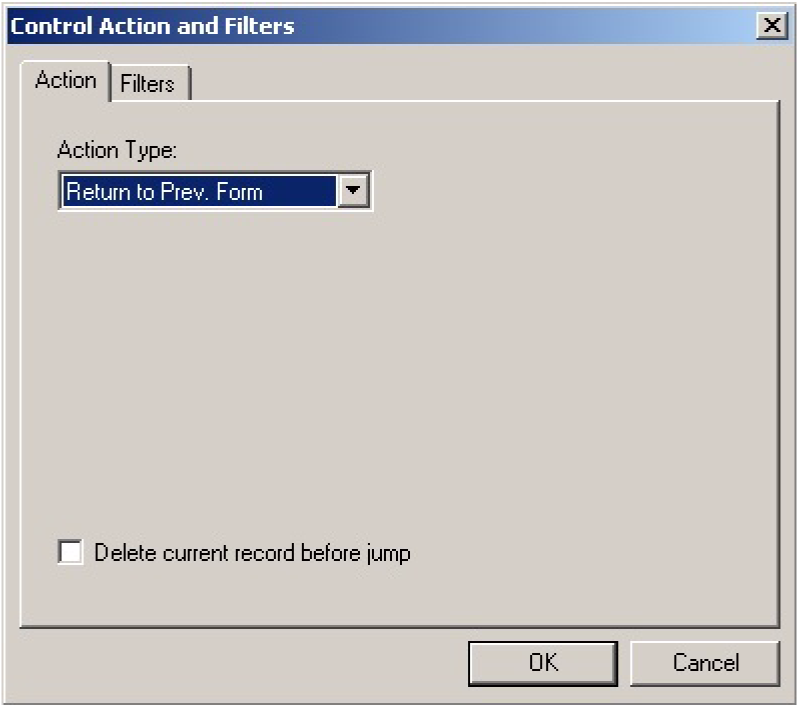
Tip You can also select a control by clicking its icon on the Workspace palette.

2 Click the **Edit Action** button to open the Control Action and Filters dialog box.

3 Select **Return to Prev. Form** from the Action Type list.

The Return to Prev. Form action specifies that when you tap this button on the handheld device, the current form closes and the previously displayed form appears. The Control Action and Filters dialog box should now look like the example shown in the following figure:

Figure 4.19 Control Action and Filters dialog box



Action Type: Return to Prev. Form

□ Delete current record before jump

4 Click the **OK** button to close the Control Action and Filters dialog box and apply the selected action.

The Notes form is now complete.

5 Close the Notes form.

6 Open the Main form by double-clicking the **Main form icon** on the UI tab of the Workspace palette.

7 Click the **Notes** button to display its properties in the Property space palette and then click the **Edit Action** button.

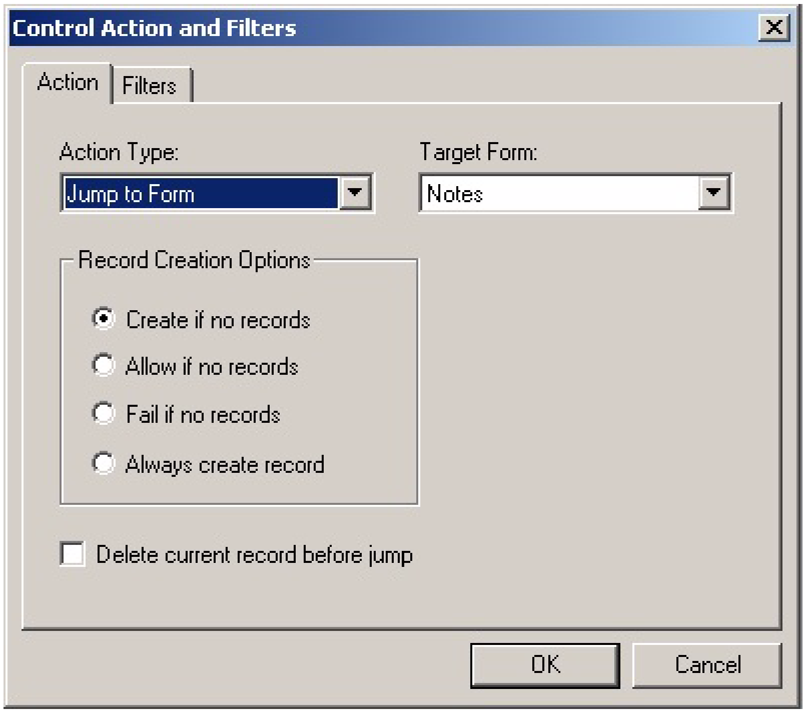
The Control Action and Filters dialog box appears.

8 Change the settings as follows:

- **Action Type:** Select **Jump to Form**.
- **Target Form:** Select **Notes**.
- **Record Creation Options:** Use the default setting, **Create if no records**.

When you have finished, the dialog box should look like the example shown in the following figure:

Figure 4.20 Control Action and Filters dialog box for Notes button



Control Action and Filters

Action Type: Jump to Form

Target Form: Notes

Record Creation Options

● Create if no records
○ Allow if no records
○ Fail if no records
○ Always create record

□ Delete current record before jump

9 Click the **OK** button to close the Control Action and Filters dialog box.

10 Close the Main form.

11 Save the application and then continue with the following section.

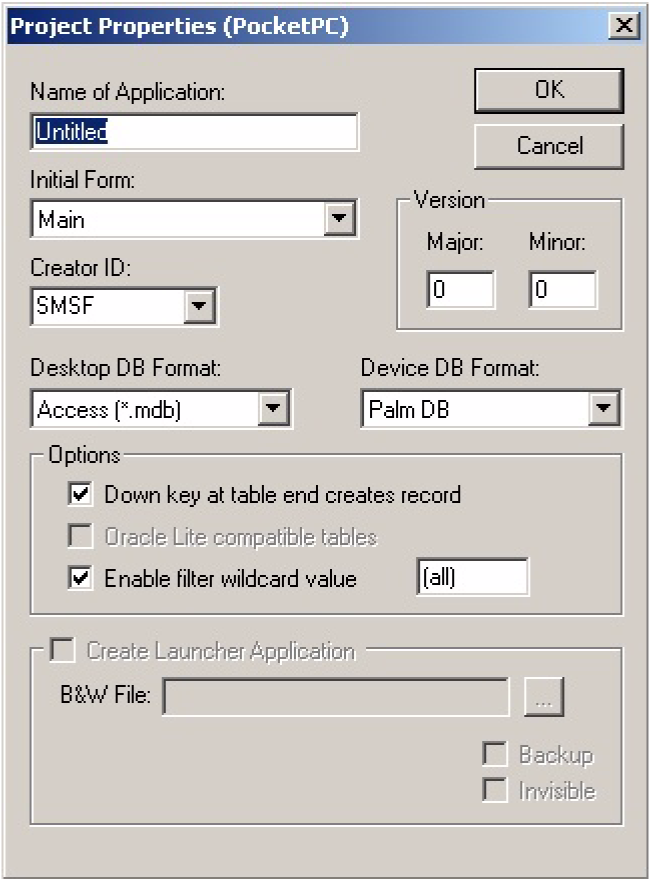
Step 6. Setting project properties

In this step, you set the project properties for the application. Application properties include the name of the application, which form appears when the application is opened, and which database formats to use.

1 Select **Edit > Project Properties** from the menu.

- The Project Properties dialog box appears, as shown in the following figure:

Figure 4.21 Project Properties dialog box



Name of Application: Untitled

Initial Form: Main

Creator ID: SMSF

Version
Major: 0
Minor: 0

Desktop DB Format: Access (*.mdb)

Device DB Format: Palm DB

Options
☑ Down key at table end creates record
□ Oracle Lite compatible tables
☑ Enable filter wildcard value (all)

□ Create Launcher Application
B&W File:
□ Backup
□ Invisible

2 Edit the following project properties:

- **Name of Application:** type `Customers to Visit`.

  This is the name of the handheld application.

- **Initial Form:** Use the default, **Main**.

  The Initial Form setting determines which form appears first when the application runs on the handheld.

- **Creator ID**: Use the default, **SMSF**.

- **Desktop DB Format:** Use the default value.

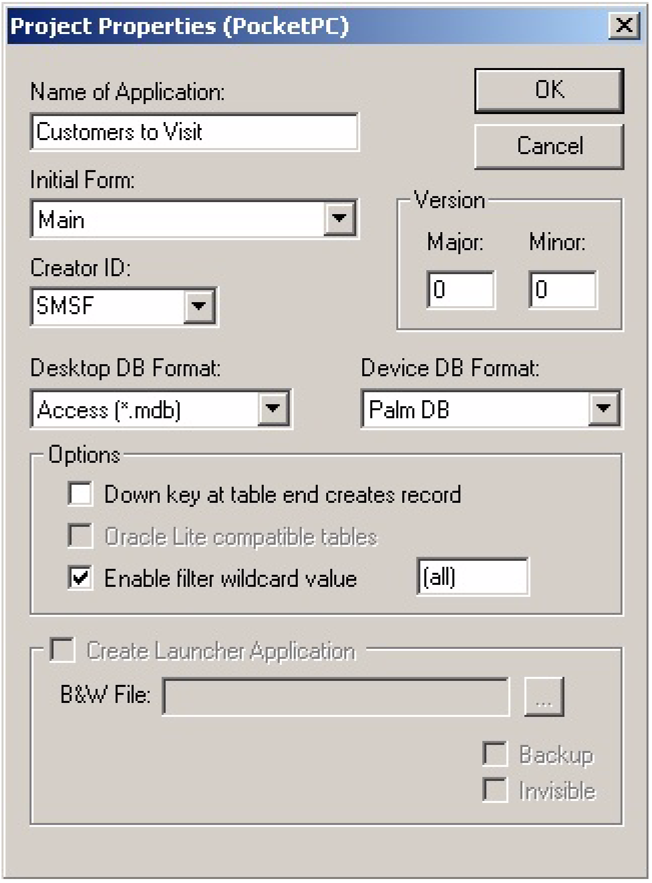
- **Device DB Format:** Use the default value.

- Under Options, clear the **Down Key at Table End Creates Record** check box and use the default settings for the other check boxes.

- *For a PalmOS target, use the same Project Properties settings as shown here for the PocketPC target.*

The Project Properties dialog box should now look like the example shown in the following figure:

Figure 4.22 Project Properties dialog box



Project Properties (PocketPC)

Name of Application: Customers to Visit

Initial Form: Main

Creator ID: SMSF

Version — Major: 0 Minor: 0

Desktop DB Format: Access (*.mdb)

Device DB Format: Palm DB

Options

- ☐ Down key at table end creates record
- ☐ Oracle Lite compatible tables
- ☑ Enable filter wildcard value (all)

☐ Create Launcher Application

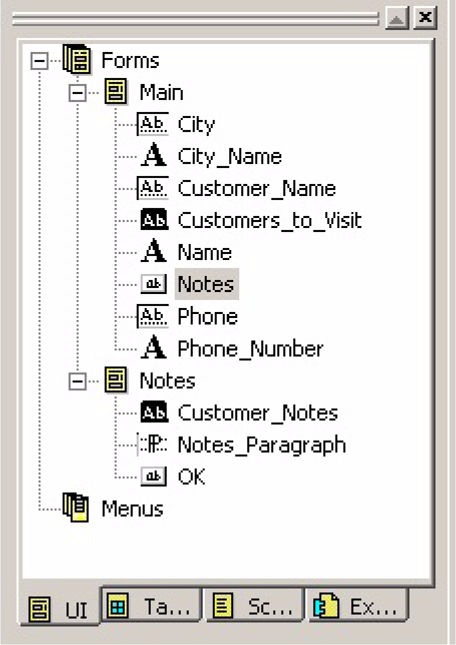
B&W File:

- ☐ Backup
- ☐ Invisible

3 Click **OK** to close the dialog box.

The Customers application is now complete. The UI tab of the Workspace palette should now look like the example shown in the following figure:

Figure 4.23 Workspace palette, UI tab for completed Customers to Visit application



The icons in the Workspace palette represent the two forms and their controls. Information about the table appears on the Tables tab. These objects make up the Customers to Visit application, which is now ready to be downloaded to your handheld device for testing.

Step 7. Downloading the application to a handheld and testing

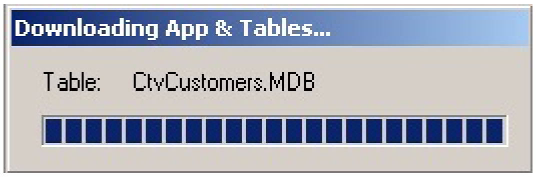
1 To test this application, place the handheld device in its cradle, or connect its cable.

2 Select **Build > Rebuild All** from the MobileApp Designer menu, click the **Rebuild All** button on the Misc toolbar, or press **F7** to build the Customers to Visit application.

3 Select **Handheld > Download App & Tables...** or click the **Download App & Tables** button on the Misc toolbar, or press **F5**.

The Download Application to Handheld progress indicator appears, as shown in the following figure:

Figure 4.24 Download Application to Handheld progress indicator



Downloading App & Tables...

Table: CtvCustomers.MDB

Tip The PalmOS download process will differ slightly than the PocketPC process shown here. A Download Application prompt will appear, asking you to press the HotSync button on the Palm cradle to begin.

4 When the transfer is complete, remove the handheld from the cradle, tap the **Start > Programs** button, and then tap the **Satellite Forms** icon.

The Satellite Forms Engine starts.

5 From the Open Application list, tap **Customers to Visit**.

The Customers to Visit screen, which is the Main form, appears and the first record from the CtvCustomers table, Mohammed's, is listed.

6 Verify that Rochester appears under City and 315-555-4393 appears under Phone.

7 Tap the **Notes** button.

8 The Customer Notes screen, which is the Notes form, opens. Check inventory is displayed.

9 Tap the **OK** button to return to the Customers to Visit screen.

10 To display the next record, press the **Down** scroll button on the handheld. Check the information displayed to see if it matches what you entered. Tap the **Notes** button to verify the note text. Tap the **OK** button to return to the Customers to Visit screen.

11 Repeat this process for the remaining three records: The Outrigger, The Fortune, and Joe's Diner.

Editing a record

1 Tap the **Menu** button on the handheld's screen and select **Records > Goto First** from the menu to return to the first record.

2 Tap the **Notes** button to open the Notes form.

3 Select the text Check inventory by tapping after the **y** in **inventory** and dragging the stylus to the left (highlighting the text as you drag).

4 Now write `Get RMA number` in the pen input writing area, or by using the device's keyboard if applicable, or use the onscreen keyboard.

The new text replaces the original text on the screen.

5 Tap the **OK** button to return to the Customers to Visit screen and then select **Options > Exit** from the handheld's menu.

Step 8. Uploading changes and verifying

The last step in this quick tour is to upload the Customers to Visit application back to your development computer and verify the changes made on your handheld device.

1 Place the handheld in its cradle, or connect its cable.

2 With the Customers application open in MobileApp Designer, select **Handheld > Upload Tables** from the menu or click the **Upload Tables** button on the Misc toolbar.

Tip The PalmOS upload process will prompt you to press the HotSync button on the Palm cradle to begin.

3 When the transfer is complete, open the **CtvCustomers** table in MobileApp Designer and verify that `Get RMA number` appears in the NOTES column for the first record.

Conclusion

Congratulations! You have now created and tested your first Satellite Forms application. At this point, you should have a good understanding of the basic concepts involved in creating an application using MobileApp Designer.

The next chapter, MobileApp Designer Reference, on page 67, provides additional details on how to work with MobileApp Designer.

Using Actions, Filters, Extensions, and Color, on page 177, provides instructions on setting control actions and filters, and introduces extensions.

Integrating with your Database, on page 195, explains the basics of integrating a Satellite Forms application with a database application.

The basic knowledge you gained from the tutorial you just completed should make the information contained in these chapters easier to understand and use.

For a more complex sample that focuses on demonstrating the capabilities of Satellite Forms rather than on the systematic construction of an application, see Sample Application: Work Order, on page 527.

Chapter 5
MobileApp Designer Reference

This chapter presents a complete reference for using MobileApp Designer to design, assemble, and build custom applications. It is organized into the following sections:

- MobileApp Designer main window: A description of the appearance and function of MobileApp Designer's user-interface components.
- Project Properties dialog box: Instructions on how to set the project properties available in MobileApp Designer.

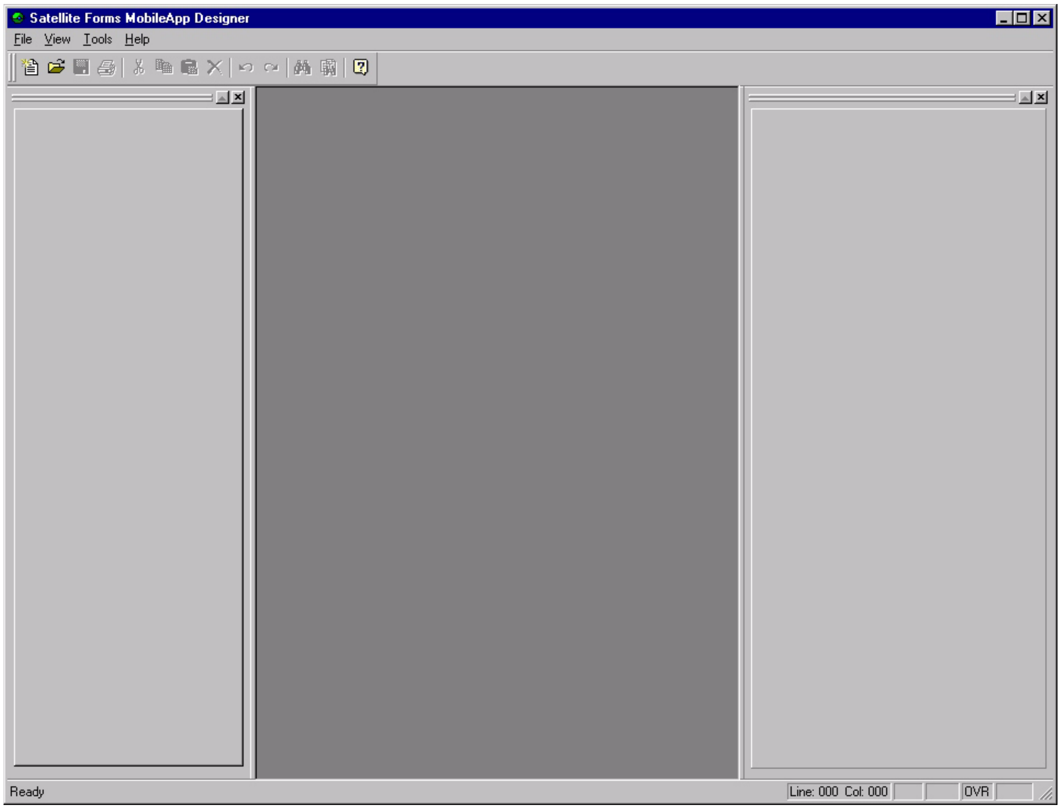
MobileApp Designer main window

MobileApp Designer is the desktop computer tool you use to create the forms and tables that make up Satellite Forms applications. It has six basic elements: the main menu, the toolbars, the Workspace palette, the Propertyspace palette, the status bar, and the desktop.

To run MobileApp Designer, click the **Start** button on the Windows Taskbar, then select **Programs > Satellite Forms 8.0**, and click **MobileApp Designer**.

When you first run MobileApp Designer, a blank desktop is displayed as shown in the following figure:

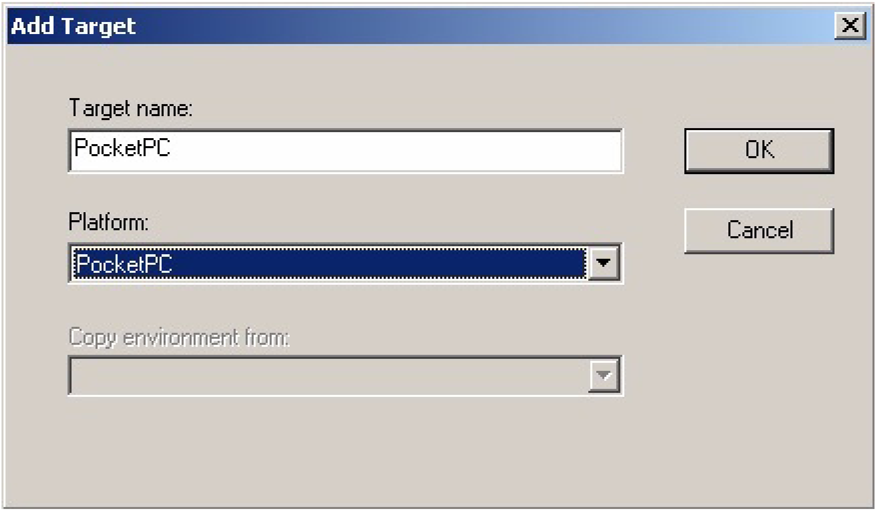
Figure 5.1 MobileApp Designer main window



To create a new project, click the **New Project** button on the General toolbar, select **File > New Project** from MobileApp Designer menu, or use the keyboard shortcut, **Ctrl** + **N**.

The **Add Target** dialog box appears as shown in the following figure:

Figure 5.2 Add Target dialog box



Add Target

Target name: PocketPC

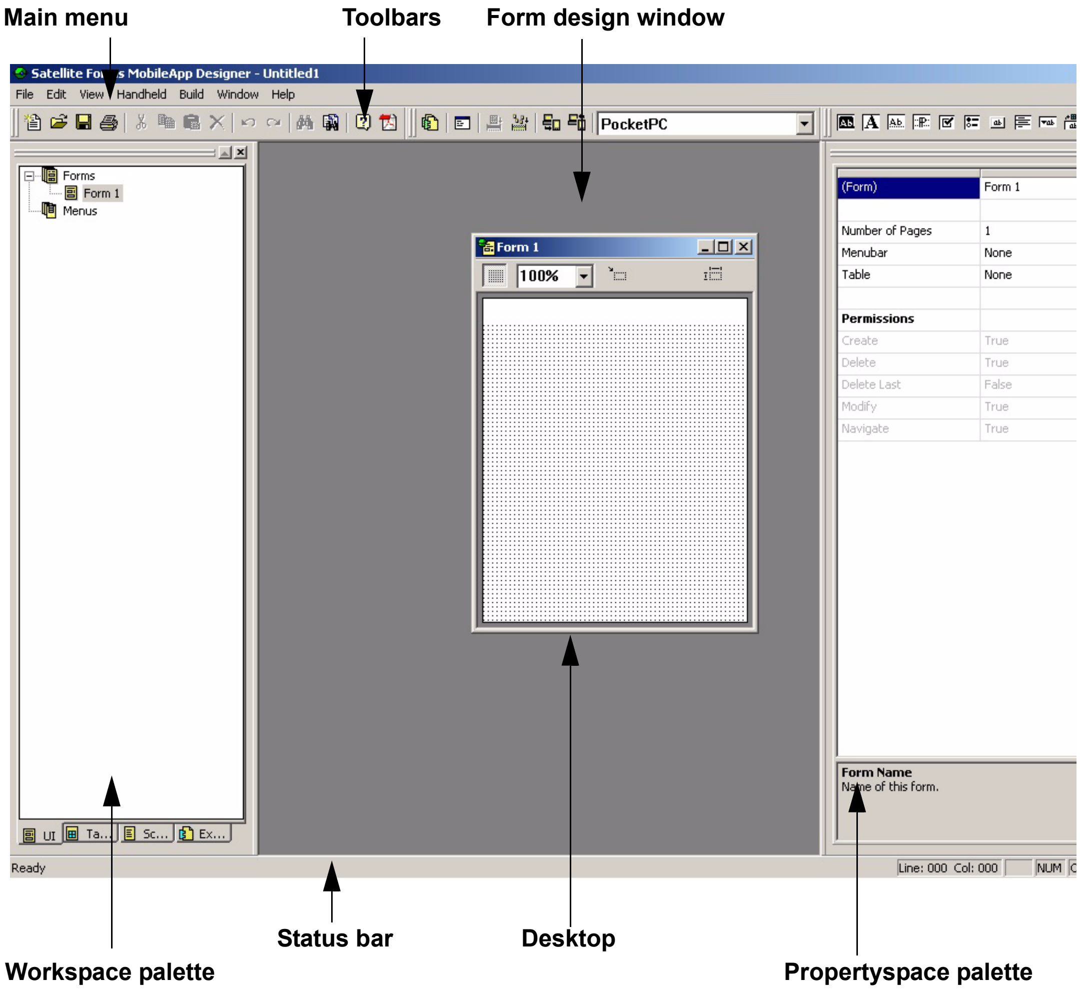
Platform: PocketPC

Copy environment from:

A project can contain one or more application targets, each of which can have a distinct name, if desired. An application target is a specific set of configurations for a handheld device and OS. For example, you can create a project with application targets for Palm and Pocket PC handhelds. You can then build the project for one application target at a time or for both application targets simultaneously.

When you open a new project, MobileApp Designer appears as shown in the following figure:

Figure 5.3 MobileApp Designer window with new project open



(Form)	Form 1
Number of Pages	1
Menubar	None
Table	None
Permissions	
Create	True
Delete	True
Delete Last	False
Modify	True
Navigate	True

Form Name
Name of this form.

- The Workspace palette appears in the left pane.
- The desktop appears in the center of the MobileApp Designer window. The Form design window with a default form loaded appears on the desktop when you first open a new project.
- The Propertyspace palette appears in the right pane.
- The project name appears in the title bar.

Workspace palette

The Workspace palette displays the contents of the current project in a tabbed, dockable pane. The Workspace palette contains four tabs: UI, Tables, Scripts, and Extensions. These tabs organize all of the corresponding items in your project.

Each tab contains one or more icons that organize the contents of the current project. If an icon contains other items, a plus sign (+) appears to the left of the icon. To expand the list of items under an icon, click the plus sign. To collapse the list of items under an icon, click the minus sign (-) to the left of the icon.

To dock the Workspace palette, drag the window to the desired edge of the MobileApp Designer main window and drop it there. To undock the Workspace palette, drag the top of the palette to the center of the desktop and drop it there. The Workspace then palette behaves like an ordinary window on the desktop.

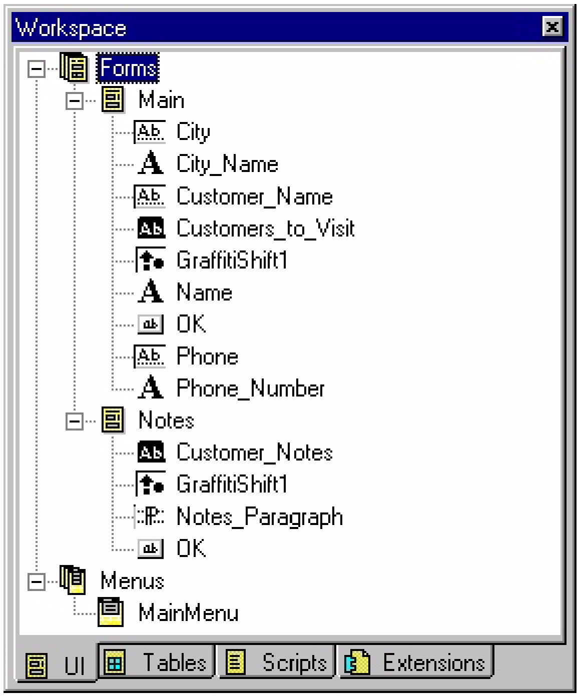
To set the size of the Workspace palette while it is docked, drag the side adjacent to the desktop as desired. To set the size of the Workspace palette while it is undocked, drag the desired edge or edges until it is the right size.

To view or hide the Workspace palette, whether it is docked or not, select **View > Project Contents** from the MobileApp Designer menu.

Each tab is shown and described in this section in order of appearance.

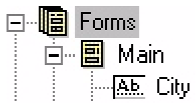
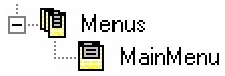
UI tab The UI tab organizes all user interface elements in the current project. The undocked Workspace palette with the UI tab selected is shown in the following figure:

Figure 5.4 Workspace palette (undocked), UI tab



The root icons on the UI tab shown above are defined in the following table:

Table 5.1 UI tab root icons

Icon	Description
	The **Forms** icon shows all forms in the current project, sorted alphabetically. Any controls in a form are listed as sub-branches under the form's icon, under Main and Notes in this case.
	The **Menus** icon shows all menus in the current project.

To perform a variety of actions on the items in the UI tab, use the actions listed in the following table:

Table 5.2 Workspace palette, UI tab mouse actions

Action	Effect
Double-click the **Forms** root icon.	Displays the Project Properties dialog box, as shown in Figure 5.13 on page 83, which allows you to set a variety of properties for the current application target.
Right-click the **Forms** root icon.	Displays a pop-up menu. • Select **Insert Form** to add a new form in the current project. • Select **Paste Form** to paste a form from the clipboard into the current project.
Double-click a **Form** icon.	Opens the form on the desktop and displays its properties on the Propertyspace palette.
Right-click a **Form** icon.	Displays a pop-up menu: • Select **Cut Form** to remove the form from the current project and place it on the clipboard. • Select **Delete Form** to remove the form from the current project. • Select **Copy Form** to copy the form to the clipboard. • Select **Paste Form** to paste a form from the clipboard into the current project. • Select **Paste Control** to paste a control from the clipboard onto the selected form. This option appears only if the item on the clipboard is a control. **Note**: MobileApp Designer prompts you to confirm both the **Cut Form** and **Delete Form** operations before it either cuts or deletes the form.
Click the plus or minus sign to the left of a **Form** icon.	Displays or hides the form's control icons.
Double-click a **Control** icon.	Selects the control and displays its properties in the Propertyspace palette.
Right-click a **Control** icon.	Displays a pop-up menu: • Select **Control Script** to jump to the Scripts tab and display the control's script on the desktop. This menu option is only available when the selected control's action is set to Run Script • Select **Cut Control** to remove the form from the current project and place it on the clipboard. • Select **Delete Control** to delete the control. MobileApp Designer does not prompt you before deleting the selected control. • Select **Copy Control** to copy a control to the clipboard.
Right-click a grayed-out **Control** icon	Displays a pop-up menu: • Select **Include in Target** to add the selected control to the current application target. **Note:** A Control icon is grayed out when the control is used on the same form in a different application target, but not in the current application target.

Table 5.2 Workspace palette, UI tab mouse actions *(Continued)*

Action	Effect
Right-click the **Menus** root icon	Displays a pop-up menu: • Select **Insert Menubar** to add a new Menubar to the menu. • Select **Paste** to paste a Menubar from the clipboard into the current project.
Click the plus or minus sign to the left of a **Menus** icon.	Displays or hides the menu's Menubar icons.
Click a **Menubar** icon	Selects the Menubar and displays its properties in the Propertyspace palette.
Right-click a **Menubar** icon	Displays a pop-up menu: • Select **Insert Menu** to add a new menu to the selected Menubar. • Select **Cut Menubar** to remove the Menubar from the current project and place it on the clipboard. • Select **Delete Menubar** to remove the Menubar from the current project. MobileApp Designer does not prompt you before deleting the selected Menubar. • Select **Copy Menubar** to copy the Menubar to the clipboard. • Select **Paste Menu** to paste a menu from the clipboard into the current project.
Click a **Menu** icon	Selects the menu and displays its properties in the Propertyspace palette.
Right-click a **Menu** icon	Displays a pop-up menu: • Select **Insert Menu Item** to add a new menu item to the selected menu. • Select **Cut Menu** to remove the menu from the current project and place it on the clipboard. • Select **Delete Menu** to remove the menu from the current project. MobileApp Designer does not prompt you before deleting the selected menu. • Select **Copy Menu** to copy the menu to the clipboard. • Select **Paste Menu Item** to paste a menu from the clipboard into the current project.
Click the plus or minus sign to the left of a **Menubar** icon.	Displays or hides the Menubar's Menu Items icons.
Click a **Menu Item** icon	Selects the Menu Item and displays its properties in the Propertyspace palette.
Right-click a **Menu Item** icon	Displays a pop-up menu: • Select **Cut Menu Item** to remove the Menu Item from the current project and place it on the clipboard. • Select **Delete Menu Item** to remove the Menu Item from the current project. MobileApp Designer does not prompt you before deleting the selected Menu Item. • Select **Copy Menu Item** to copy the Menu Item to the clipboard.

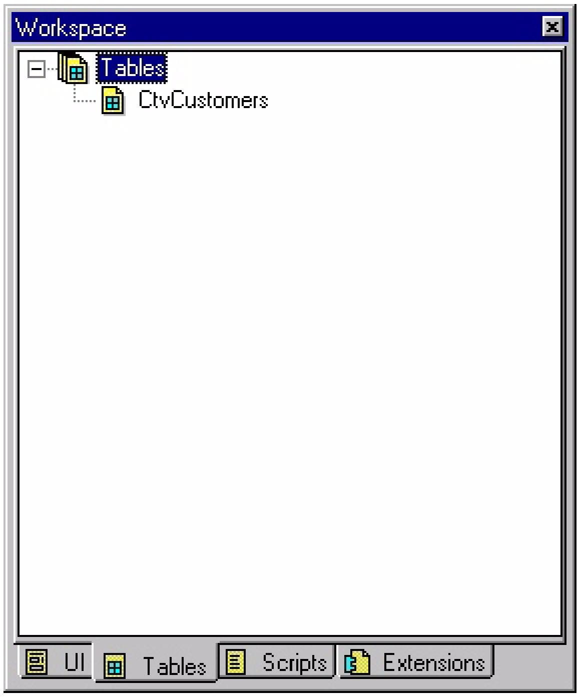
Table 5.2 Workspace palette, UI tab mouse actions *(Continued)*

Action	Effect
Right-click a grayed-out **Menubar**, **Menu**, or **Menu Item** icon	Displays a pop-up menu: • Select **Include in Target** to add the selected Menubar, Menu, or Menu Item to the current application target. **Note:** An icon is grayed out when the item is used on the same form in a different application target, but not in the current application target.

Tables tab

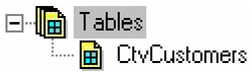
The Tables tab organizes all tables in the current project. The undocked Workspace palette with the Tables tab selected is shown in the following figure:

Figure 5.5 Workspace palette (undocked), Tables tab



The root icon on the Tables tab shown above is defined in the following table:

Table 5.3 Tables tab root icon

Icon	Description
	The **Tables** icon shows all tables in the current project, sorted alphabetically.

To perform a variety of actions on the items in the Tables tab, use the actions listed in the following table:

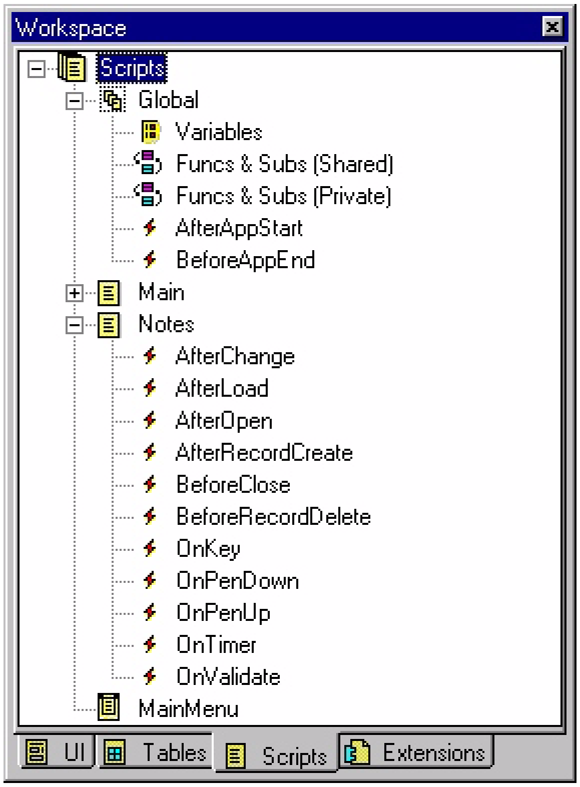
Table 5.4 Workspace palette, Tables tab mouse actions

Action	Effect
Right-click the **Tables** root icon.	Displays a pop-up menu: • Select **Insert Table** to open a new table on the desktop. • Select **Import Table** to import a table schema from an external database. • Select **Paste Table** to paste a table from the clipboard into the current project. This option is available only if the item on the clipboard is a table.
Click a **Table** icon.	Opens the Table design dialog box for the selected table.
Right-click a **Table** icon.	Displays a pop-up menu: • Select **Cut Table** to remove the table from the current project and place it on the clipboard. • Select **Delete Table** to delete the table. • Select **Copy Table** to copy the table to the clipboard. • Select **Paste Table** to paste a table from the clipboard into the current project. This option is available only if the item on the clipboard is a table. **Note**: MobileApp Designer prompts you to confirm both the **Cut Table** and **Delete Table** operations before it either cuts or deletes the table.

Scripts tab

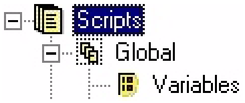
The Scripts tab organizes all scripts in the current project. The undocked Workspace palette with the Scripts tab selected is shown in the following figure:

Figure 5.6 Workspace palette (undocked), Scripts tab



The root icon on the Scripts tab shown above is defined in the following table:

Table 5.5 Scripts tab root icon

Icon	Description
	The **Scripts** icon shows all scripts in the current project. Global scripts apply to the entire project, while form and menu scripts apply only to the form or menu that owns them.

To perform a variety of actions on the items in the Scripts tab, use the actions listed in the following table:

Table 5.6 Workspace palette, Scripts tab mouse actions

Action	Effect
Right-click the **Scripts** root icon, a **Form** or **Menu** icon.	Displays a pop-up menu: • Select **Show Script** to open the Global/Variables script in the script editing window.
Double-click a **Form** or **Menu** icon.	Opens or brings to the front the selected form or menu.
Right-click an individual **Script** icon, AfterChange, AfterLoad, etc.	Displays a pop-up menu: • Select **Compile Script** to compile the selected script.

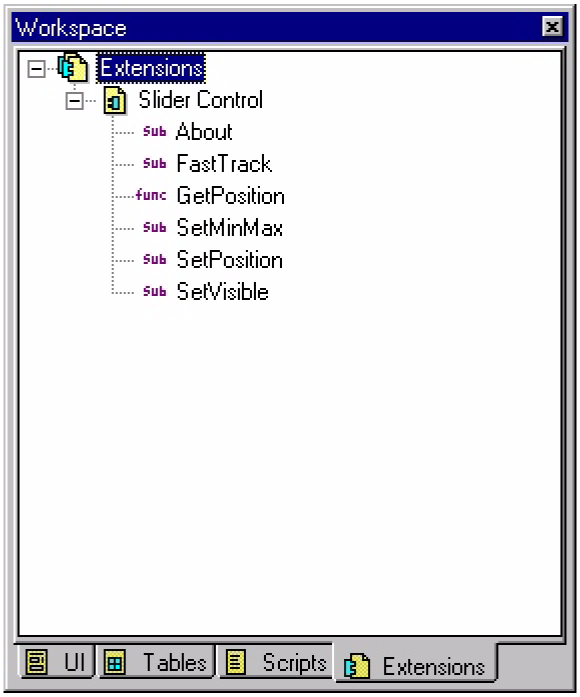
Table 5.6 Workspace palette, Scripts tab mouse actions *(Continued)*

Action	**Effect**
Click an individual **Script** icon, AfterChange, AfterLoad, etc.	Opens the selected script in the script editing window.

Extensions tab

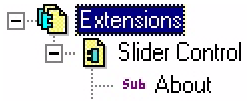
The Extensions tab organizes all extensions in the current project. The undocked Workspace palette with the Extensions tab selected is shown in the following figure:

Figure 5.7 Workspace palette (undocked), Extensions tab



Extensions are programs that provide added functionality to your applications. See Adding extensions to Satellite Forms on page 188 for more information.

Table 5.7 Extensions tab root icon

Icon	**Description**
	The **Extensions** icon shows all extensions in the current project, sorted alphabetically.

To perform a variety of actions on the items in the Extensions tab, use the actions listed in the following table:

Table 5.8 Workspace palette, Extensions tab mouse actions

Action	Effect
Right-click the **Extensions** root icon.	Displays a pop-up menu: • Select **Extensions...** to display the Available Extensions dialog box, as shown in Figure 5.15 on page 86, from which you can select extensions to add or remove by checking or clearing the check box to the left of each extension name.
Double-click the **Extensions** icon.	Displays the Available Extensions dialog box, as shown in Figure 5.15 on page 86, from which you can select extensions to add or remove by checking or clearing the check box to the left of each extension name.
Right-click an **Extension** icon.	Displays a pop-up menu. • Select **Properties...** to display the Extension Properties dialog box, which provides details about the selected extension. • Select **Delete** to remove the extension from the current project. **Note**: MobileApp Designer prompts you to confirm the **Delete** operation before it removes the selected extension from the current project.
Double-click an **Extension** icon.	Displays the Extension Properties dialog box, which provides details about the selected extension.
Right-click an extension **Method** icon.	Displays a pop-up menu. • Select **Properties...** to display the Properties of Method dialog box, which provides details about the selected method.
Double-click an extension **Method** icon.	Displays the Properties of Method dialog box, which provides details about the selected method.

MobileApp Designer menus

The MobileApp Designer menu bar provides menus that help you complete specific tasks. The **File**, **View**, **Tools**, and **Help** menus are always available. When you open a project with MobileApp Designer, the **Edit**, **Handheld**, **Build**, and **Window** menus appear on the menu bar.

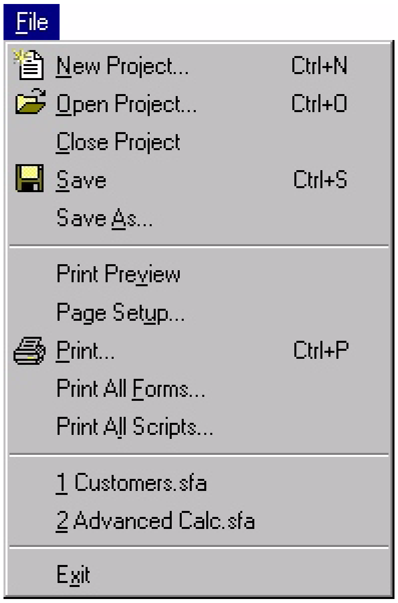
Some menu options have keyboard shortcuts. For example, pressing **Ctrl + S** on the keyboard is the same as selecting **File** > **Save** from the menu. If a keyboard shortcut for an option exists, it appears next to the menu option, as shown in Figure 5.8 on page 79.

The following sections describe the options on each menu.

MobileApp Designer File menu

The **File** menu, as shown in the following figure, provides basic file management features for creating, opening, and saving project files.

Figure 5.8 MobileApp Designer File menu



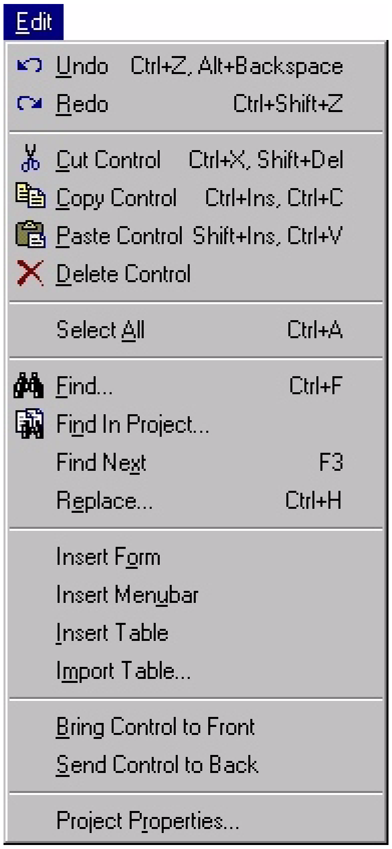
The **File** menu provides the following options:

- **New Project...:** Opens a new Satellite Forms project.
- **Open Project...:** Displays a standard file **Open** dialog box that allows you to open an existing project.
- **Close Project:** Closes the current project. If you have made unsaved changes to the project, MobileApp Designer prompts you to save the project before closing it.
- **Save:** Saves the current project. If the current project is new and you have not saved it, MobileApp Designer opens the **Save As** dialog box, which allows you to name the project and select the directory in which to save it.
- **Save As...:** Opens the **Save As** dialog box, which allows you to rename the project and select the directory in which to save it.
- **Print Preview:** Opens a **Print Preview** window on the desktop that shows you how your printed pages will appear.
- **Page Setup...:** Displays the standard **Print Setup** dialog box, which allows you to select a printer and set other printing options.
- **Print...:** Displays the standard **Print** dialog box, which allows you to set printing options and to print the current form.
- **Print All Forms...:** Displays the standard **Print** dialog box, which allows you to set printing options and to print all forms in the current project.
- **Print All Scripts...:** Displays the standard **Print** dialog box, which allows you to set printing options and to print all scripts in the current project.
- **Recent Project List:** Displays up to the four most recently opened projects. Click the desired project file name or type the corresponding number on the keyboard to open the selected project.
- **Exit:** Closes MobileApp Designer. If you have a project open that has unsaved changes, MobileApp Designer prompts you to save the project before closing.

MobileApp Designer Edit menu

The wording of the options on the **Edit** menu and in some cases the options that appear on the menu, depend on how you are using MobileApp Designer. Any options that are not available for the current operation are either grayed-out or do not appear on the **Edit** menu. A representative Edit menu appears in the following figure:

Figure 5.9 MobileApp Designer Edit menu

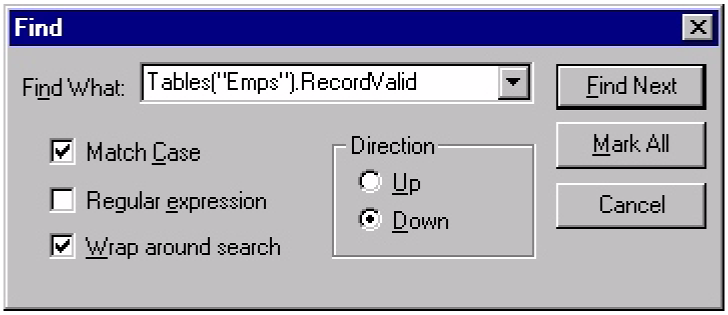


The **Edit** menu contains some or all of the following options:

- **Undo:** Select this option to reverse the preceding action. For example, select this option to restore the most recent deletion from a script. This option is available only if the preceding action can be undone.
- **Redo:** Select this option to repeat the preceding action. For example, select this option to restore the portion of a script removed with the Undo option. This option is available only if the preceding action can be re-done.
- **Cut <*Item*>:** Cuts the selected item from the project and places it on the clipboard. If you select multiple items, all are cut to the clipboard. You can paste items from the clipboard into the same project or into a different project. In some cases, the type of item that can be cut is displayed in the menu option text.
- **Copy <*Item*>:** Copies the selected item and places it on the clipboard. If you select multiple items, all are copied to the clipboard. You can paste items from the clipboard into the same project or into a different project. In some cases, the type of item that can be copied is displayed in the menu option text.

- **Paste <*Item*>:** Pastes the item from the clipboard into the current project. If the clipboard contains multiple items, all are pasted into the current project. You can paste items from the clipboard into the same project or into a different project. In some cases, the type of item currently on the clipboard is displayed in the menu option text.
- **Delete <*Item*>:** Deletes the selected item from the project. Some items, such as scripts, allow you to select Undo to restore the most recent deletion. Controls and most other items do not. If you are unsure about a deletion, use the **Edit > Cut <*Item*>** option instead. You can then restore the item by pasting it back into the form, table, or other part of the current project. Remember that the clipboard holds only one item at a time and any other Cut or Copy operation replaces the previous item in the clipboard.
- **Select All:** Selects all controls on the current form or all records in the table open in the table editor. You can then Cut, Copy, or Delete the selected items as desired.
- **Find...:** Applies only to scripts. Displays the **Find** dialog box, which allows you to specify the text to search for in the current script and other search options. An example **Find** dialog box appears in the following figure:

Figure 5.10 Find dialog box



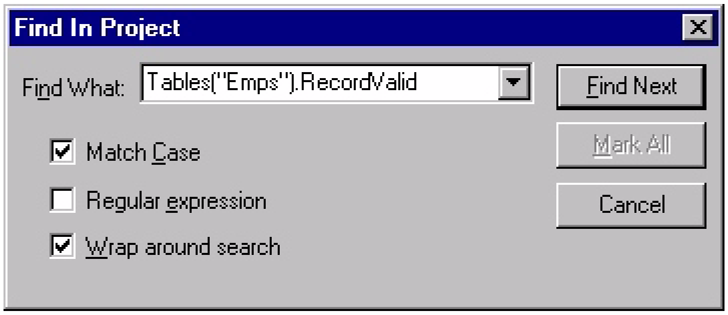
Find What: Tables("Emps").RecordValid

☑ Match Case
□ Regular expression
☑ Wrap around search

Direction
○ Up
● Down

- **Find in Project...:** Applies only to scripts. Displays the **Find in Project** dialog box, which allows you to specify the text to search for in all scripts in the current project and other search options. An example **Find in Project** dialog box appears in the following figure:

Figure 5.11 Find in Project dialog box



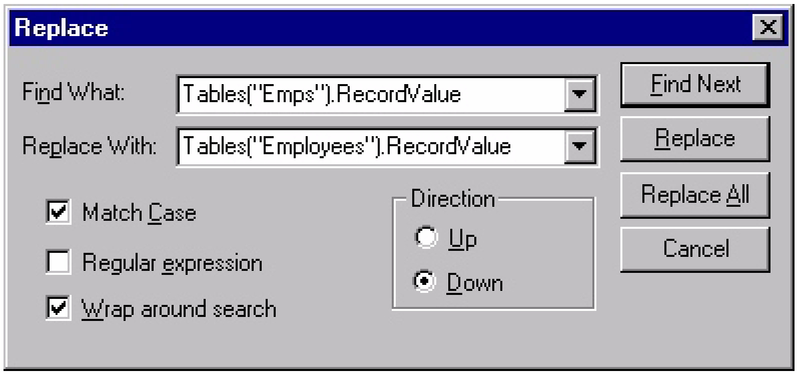
Find What: Tables("Emps").RecordValid

☑ Match Case
□ Regular expression
☑ Wrap around search

- **Find Next:** Applies only to scripts. Searches for the next occurrence of the text entered in the **Find** or **Find in Project** dialog box.
- **Replace...:** Applies only to scripts. Displays the **Replace** dialog box, which allows you to specify the text to search for, the text to replace the search text, and

other options. You can replace individual instances of the search text or replace all such instances using the buttons in the Replace dialog box. An example **Replace** dialog box appears in the following figure:

Figure 5.12 Replace dialog box



Replace

Find What: Tables("Emps").RecordValue

Replace With: Tables("Employees").RecordValue

☑ Match Case

□ Regular expression

☑ Wrap around search

Direction
○ Up
● Down

- **Insert Form:** Adds a new, blank form to the current project. See Creating and editing forms on page 114 for more information.

- **Insert Menubar:** Adds a new, empty menu to the current project. See Working with menus on page 118 for more information.

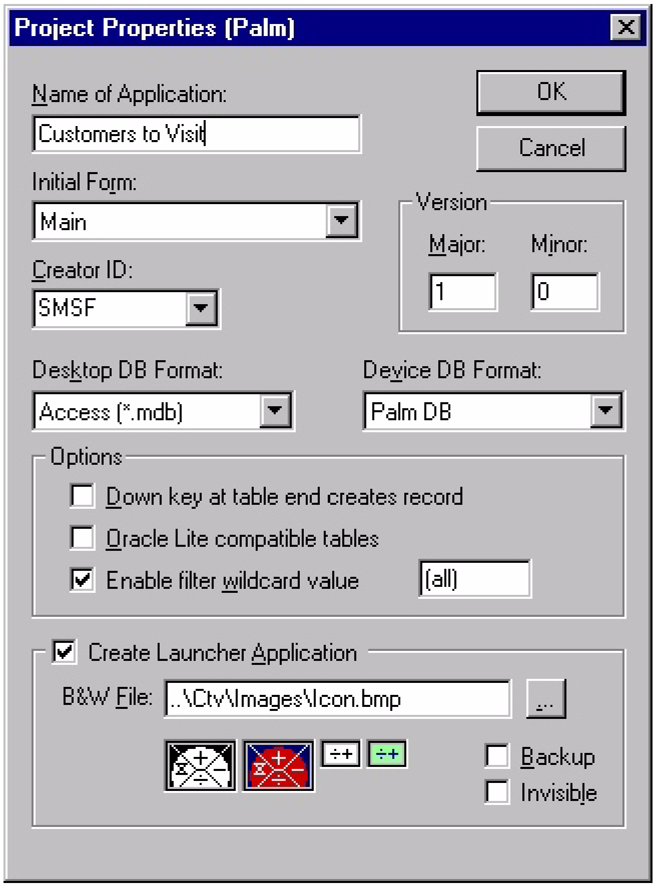
- **Insert Table:** Adds a new, empty table to the current project. See Working with tables on page 103 for more information.

- **Import Table:** Imports a table schema into the current project. Use this option to load one or more tables from an external ODBC-enabled database. MobileApp Designer automatically creates the structure (schema) of the tables, but does not import any data they may contain into the current project. See Importing tables on page 108 for more information.

- **Bring Control to Front/Send Control to Back:** A control's stacking or z-order determines whether it draws on top of or underneath another control in the same location on a form. The **Bring Control to Front** option places the selected control at the top of the z-order, making it draw on top of all other controls at the same location on the form. The **Send Control to Back** option places the selected control at the bottom of the z-order, making it draw underneath all other controls at the same location on the form.

- **Project Properties...:** Opens the **Project Properties** dialog box for the current project and application type: Palm or Pocket PC. An example **Project Properties** dialog box appears in the following figure:

Figure 5.13 Project Properties dialog box



Project Properties (Palm)

Name of Application: Customers to Visit

Initial Form: Main

Creator ID: SMSF

Version
Major: 1
Minor: 0

Desktop DB Format: Access (*.mdb)

Device DB Format: Palm DB

Options
☐ Down key at table end creates record
☐ Oracle Lite compatible tables
☑ Enable filter wildcard value (all)

☑ Create Launcher Application
B&W File: ..\Ctv\Images\Icon.bmp
☐ Backup
☐ Invisible

- **Name of Application:** The name of the current application target. This is the application name that appears on the Satellite Forms **Select Application to Run** window on the handheld device. The name can be the same or different for each target. In the example above, the target is Palm OS.
- **Initial Form:** The form that first appears when a user starts the application.
- **Version:** The Major and Minor version numbers for the application.
- **Creator ID:** The four-character unique ID of the application creator. **This option originates from and applies primarily to Palm OS applications, but also affects Pocket PC applications.** The Creator ID for your applications must be unique to avoid conflicts with other applications running on the same handheld device. Use one of the ten Creator IDs that Satellite Forms has reserved (SMS0 to SMS9), or enter your own unique Creator ID. Deployments internal to an organization can take advantage of the reserved IDs provided other applications within the organization are not already using them. Deployments to third parties should **always** have a unique Creator ID to avoid potential conflicts with other existing applications. You can obtain a unique Creator ID for free from the ACCESS Americas (formerly PalmSource) website, which maintains a database of registered Creator IDs. The Creator ID

makes up a part of the filenames for database tables in your application on both the Palm OS and Pocket PC platforms, so if you are creating a cross-platform application, you need to make sure that it is the same for all platform targets.

- **Desktop DB Format:** The database engine plug-in and format of the database on the desktop to which the application is linked.
- **Device DB Format:** The database engine plug-in and format of the database on the handheld device on which the application runs.

Tip While you can select either the Pocket PC DB (CDB) or Palm DB (PDB) device database format for Pocket PC applications, we strongly recommend using the Palm DB database format only. The Palm DB format provides numerous performance, synchronization, and reliability advantages over the Pocket PC DB format.

- **Options:** These settings control how an application works with tables:
  - **Down key at table end creates record:** Check this box to allow the user to add a new record to a table by pressing the down key on the handheld device when viewing the last record in the current table. A confirmation dialog box prompts the user to create a new record. When you clear this check box, pressing the down key after reaching the last record causes the handheld device to beep.

Tip You can create or delete a record using the handheld device's **Records** menu. For information, see Palm OS Satellite Forms application menus on page 227.

  - The option **Enable Filter Wildcard Value** allows you to set a wildcard value for filters. This value can be characters (for example, *) or keywords (all), or combinations of both. By default, filter wildcards are enabled and the wildcard value is set to (all). When filter wildcards are enabled and a filter's criterion is set to the wildcard value, the filter is removed. For more information on using wildcard values while setting filters, see Adding or editing a filter on page 185.
  - **Oracle® Lite compatible tables:** Check this box to enable seamless interoperability with Oracle Lite from Oracle Corporation. The Oracle Lite Consolidator, which runs with Oracle Lite on a desktop computer, reads Satellite Forms tables directly and bi-directionally synchronizes data changes. Use Oracle Lite on the desktop computer to store data; use the Oracle Lite Consolidator to transfer data to and from the handheld device. When the data is in Oracle Lite, transfer it to your full Oracle implementation using the Oracle replication utilities. For more information integrating Satellite Forms with Oracle Lite, see the Oracle Lite user manuals.
  - **Enable filter wildcard value:** Check this box and type the desired wildcard values into the associated text box to set a wildcard value for filters. This value can be characters, for example, * and ?, keywords, or combinations of both as indicated by the default value: **(all)**. For more information on using wildcard values when setting filters, see Using table filters on page 184.
- **Create Launcher Application:** Check this box and select the desired icon file by clicking the ... button to generate a launcher application with the selected icon. The user can then simply tap the icon to launch the application.

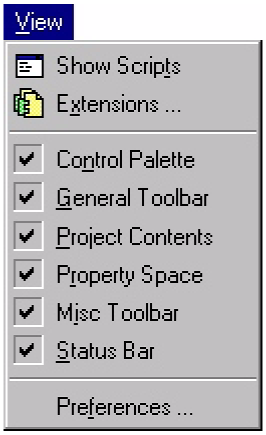
Note The option to create the launcher application applied to the PalmOS platform only in previous versions, but with Satellite Forms 8 it now applies to both the Palm OS and Pocket PC platforms. For full details, see Create and assign application icons on page 244.

- **B&W File:** Type the name of the black-and-white icon file for the launcher application in this box or click the ... button to browse for the desired icon file. The color icons will be found automatically based on a special naming convention. All icon files must be Windows.BMP format graphics files.
- **Backup:** Check this box to set the file properties so that Palm Hotsync makes a backup of the launcher application.
- **Invisible:** Check this box to make the launcher application icon invisible on the handheld device.

MobileApp Designer View menu

The **View** menu allows you to specify which tools appear in the MobileApp Designer window and to set preferences that apply to the MobileApp Designer IDE.

Figure 5.14 MobileApp Designer View menu

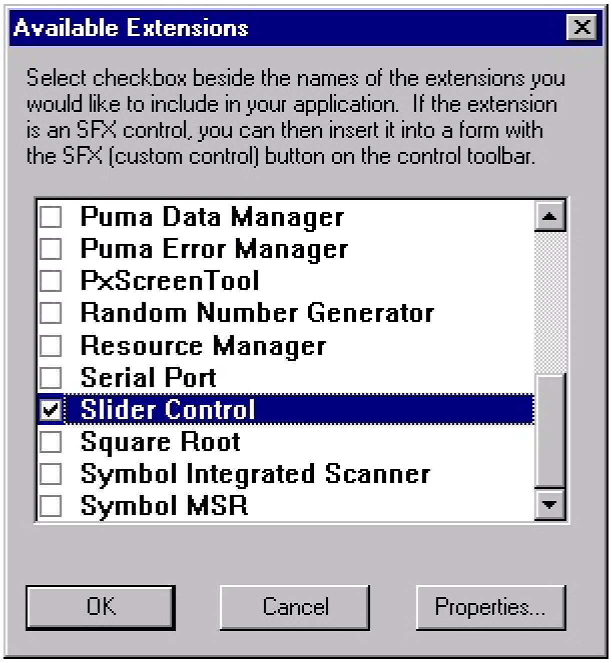


The **View** menu contains the following options:

- **Show Scripts:** Displays the Application Scripts window on the desktop with the last selected script open.
- **Extensions...:** Displays the **Available Extensions** dialog box, as shown in Figure 5.15, from which you can select the extension(s) to add to or remove from the current project by checking or clearing the check box to the left of each extension name. Extensions are programs that provide added functionality or additional controls to your applications. See Satellite Forms API Reference, on page 467 for more information on extensions.

  When you have finished adding or removing extensions, click the **OK** button to close the Available Extensions dialog box. Click the **Cancel** button to close the dialog box without making any changes to the extensions in the current project. To view information about the currently selected extension, click the **Properties...** button.

Figure 5.15 Available Extensions dialog box

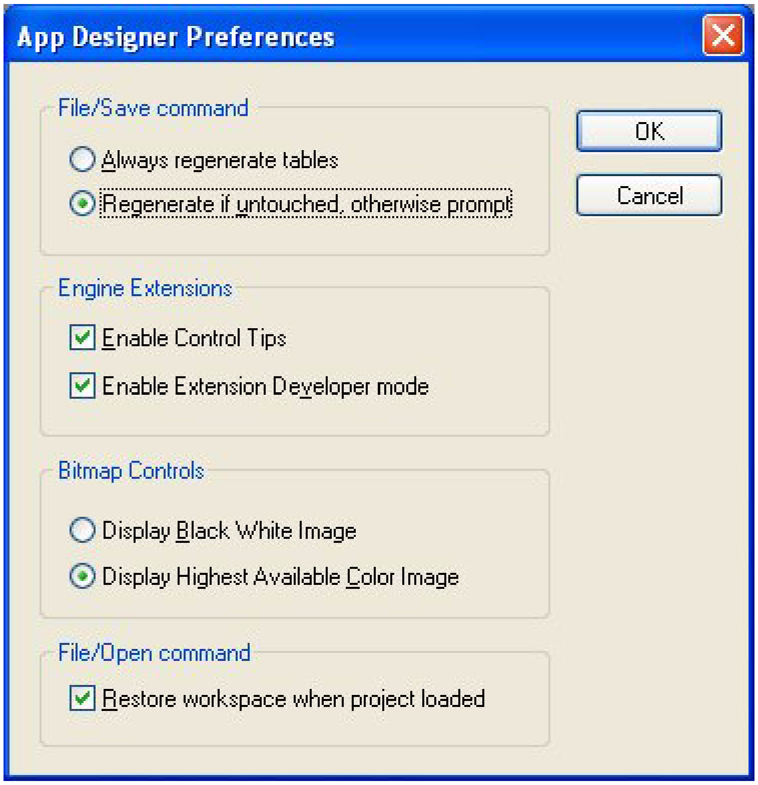


Select checkbox beside the names of the extensions you would like to include in your application. If the extension is an SFX control, you can then insert it into a form with the SFX (custom control) button on the control toolbar.

- □ Puma Data Manager
- □ Puma Error Manager
- □ PxScreenTool
- □ Random Number Generator
- □ Resource Manager
- □ Serial Port
- ☑ Slider Control
- □ Square Root
- □ Symbol Integrated Scanner
- □ Symbol MSR

- **Control Palette:** Shows/hides the Control Palette toolbar.
- **General Toolbar:** Shows/hides the General toolbar.
- **Project Contents:** Shows/hides the Workspace palette.
- **Propertyspace:** Shows/hides the Propertyspace palette.
- **Misc Toolbar:** Shows/hides the Misc toolbar.
- **Status Bar:** Shows/hides the MobileApp Designer status bar.
- **Preferences...:** Displays the MobileApp Designer Preferences dialog box, which allows you to set preferences for the MobileApp Designer IDE. An example is shown in the following figure:

Figure 5.16 MobileApp Designer Preferences dialog box



App Designer Preferences

File/Save command

○ Always regenerate tables

● Regenerate if untouched, otherwise prompt

Engine Extensions

☑ Enable Control Tips

☑ Enable Extension Developer mode

Bitmap Controls

○ Display Black White Image

● Display Highest Available Color Image

File/Open command

☑ Restore workspace when project loaded

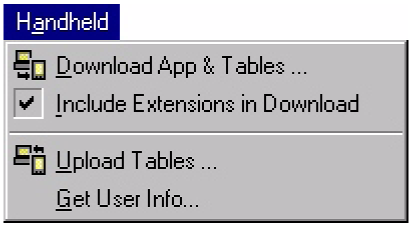
- **File/Save command:** Sets table regeneration options when you save a project. Click the desired option.
    - **Always regenerate tables:** Regenerates all tables in the current project every time you save the project, whether MobileApp Designer or another application altered the tables or not. If another application modified any of the tables in the current project, all of those changes are lost when you save the project. For this reason, this is not the recommended option.
    - **Regenerate if untouched, otherwise prompt:** Regenerates all unmodified tables in the current project, but prompts you to regenerate tables another application modified, every time you save the project. If you choose to have MobileApp Designer regenerate the tables, all of the changes the other application made to the tables are lost. This option prevents accidental loss of table schema or data changes made outside of MobileApp Designer and is the recommended option.
- **Engine Extensions:** Sets Extensions options for the MobileApp Designer IDE. Check either or both boxes.
    - **Enable Control Tips:** Displays a tooltip with information about a control on a form when you hover the mouse cursor over the control.
    - **Enable Extension Developer mode:** Sets up the MobileApp Designer IDE to develop custom C-language Extensions for Satellite Forms applications. More information on third-party extension features can be found on the Satellite Forms web site at http://www.satelliteforms.net.

- **Bitmap Controls:** Sets the display behavior of controls that contain bitmaps. Click the desired option.
  - **Display Black White Image:** Displays all bitmap controls as black-and-white (two-color) bitmaps.
  - **Display Highest Available Color Image:** Displays all bitmaps controls at the greatest color depth available. The bitmaps you attach to a control determines the maximum available color depth.
- **File/Open command:** Set options for when you open a project.
  - **Restore workspace when project loaded:** The default behaviour (when this option is checked) is to open up the form, script, and table editor windows in the MobileApp Designer desktop that were open when the project was last saved. In some rare cases, when numerous form and table editor windows were open when the project was saved and MobileApp Designer tries to open them all back up when the project is loaded, low memory errors may occur on the PC. In this instance, the best approach is to uncheck this option, so that MobileApp Designer does not automatically try to reopen all of the windows when the project is loaded. You can then save the project again with no windows open, and restore this option to the default setting if desired.

MobileApp Designer Handheld menu

The **Handheld** menu allows you to send information to or retrieve information from a handheld device.

Figure 5.17 MobileApp Designer Handheld menu



The **Handheld** menu contains the following options:

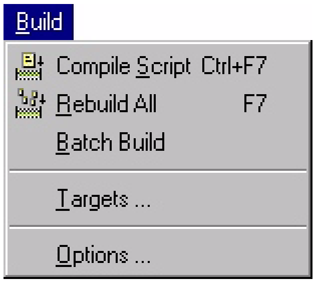
- **Download App & Tables...:** Downloads the current application and its tables to a handheld device. The **F5** hotkey is assigned to this function.
- **Include Extensions in Download:** When checked, downloads all extensions in the application to the handheld device. This menu option is only available if you have added one or more extensions to the current project. This option is checked by default if you have not downloaded the current application. Extensions only need to be downloaded one time unless you change the extension or its properties.
- **Upload Tables...:** Uploads an application's tables from the handheld device to the development computer, where MobileApp Designer can access them.
- **Get User Info...:** Retrieves and displays the User Name and User ID of the handheld device currently in the cradle (this feature applies to PalmOS devices only). The User Name is the standard HotSync User Name. See the handheld

device user manual for more information. The Satellite Forms software creates a unique numeric User ID the first time Satellite Forms synchronizes with the handheld device. The User Name and User ID are most often used by your desktop DBMS to identify the handheld currently being synchronized. See Integrating with your Database, on page 195.

MobileApp Designer Build menu

The **Build** menu allows you to compile scripts and applications and set build options for the current project.

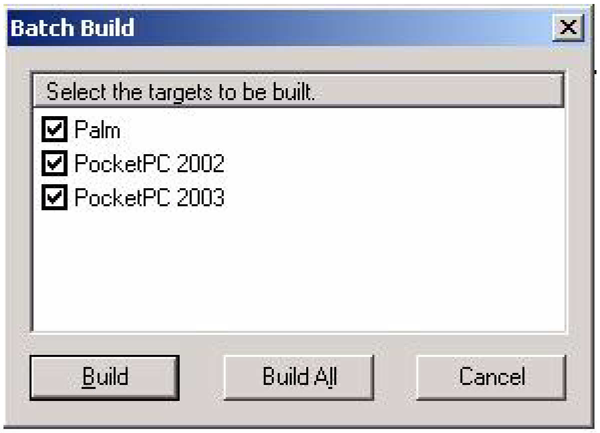
Figure 5.18 MobileApp Designer Build menu



The **Build** menu contains the following options:

- **Compile Script:** This option is available when a script is open in the Application Scripts window. Select this option or press **Ctrl + F7** to compile the open script.
- **Rebuild All:** Rebuilds the current project for the selected application target. The selected application target is shown in the Select Target combo box on the Misc toolbar, as shown in Figure 5.28 on page 97.
- **Batch Build:** Displays the Batch Build dialog box, which allows you to build one or more of the application targets defined in the current project. For example, if you have defined application targets for Palm and Pocket PC, check the box to the left of the targets you want to build and click the **Build** button. Click the **Build All** button to build all defined application targets, whether checked or not. And example Batch Build dialog box is shown in the following figure:

Figure 5.19 Batch Build dialog box

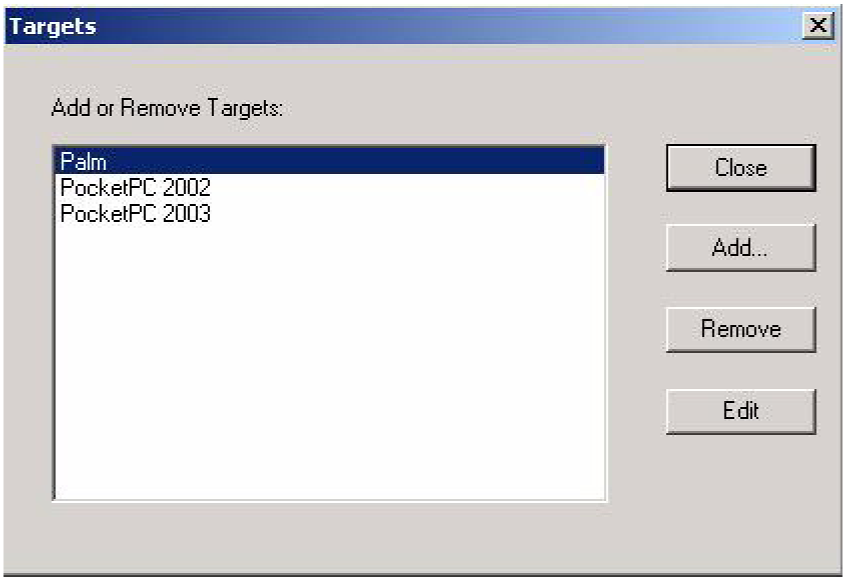


Select the targets to be built.

☑ Palm
☑ PocketPC 2002
☑ PocketPC 2003

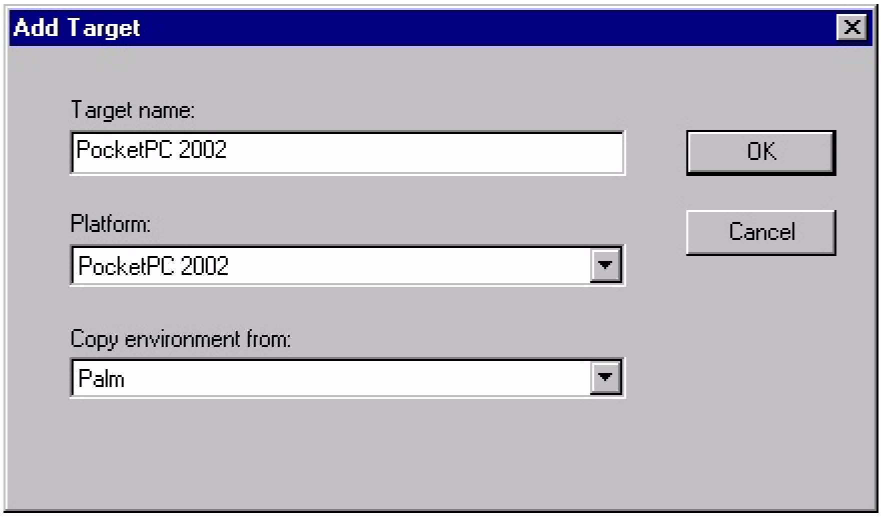
- **Targets...:** Displays the Targets dialog box, which shows the application targets defined for the current project and allows you to add, delete, or edit application targets. Each of these functions is described below. When you are finished using this dialog box, click the **Close** button. An example Targets dialog box is shown in the following figure:

Figure 5.20 Targets dialog box



- **Add...:** Click this button to add a new application target to the current project. When you add a new application target, all forms, controls, tables, and other project elements are added to the new target. All you have to do is build the new target to have an application for that platform. When you are finished using the Add Target dialog box, click the **OK** button to save all changes or click the **Cancel** button to close without saving your changes. An example of the Add Target dialog box appears in the following figure:

Figure 5.21 Add Target dialog box



Target name: PocketPC 2002

Platform: PocketPC 2002

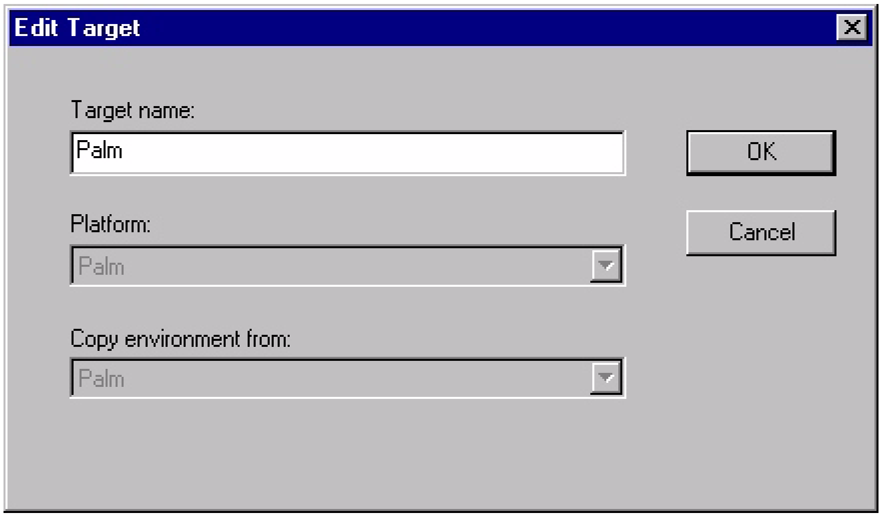
Copy environment from: Palm

- **Target Name:** Type the name of the target in this edit box. The target name cannot duplicate a name already defined in the current project. The default name is the platform name plus a numeric identifier to make it unique in the current project.

    - **Platform:** Select the desired handheld device platform from the drop list. Only platforms for which MobileApp Designer can build applications appear in this list.
    - **Copy environment from:** Select the platform type from which to copy the environment for the new application target. Only platforms for which MobileApp Designer can build applications appear in this list.

- **Remove:** Removes the target selected in the Targets dialog box from the current project. Mobile AppDesigner prompts you before removing the selected target.
- **Edit:** Allows you to edit the name of the target selected in the Targets dialog box. Changing the name of the target does not change the platform or any other property of the target. When you are finished editing the target name, click the **OK** button to save all changes or click the **Cancel** button to close without saving your changes. An example Edit Target dialog box appears in the following figure:

Figure 5.22 Edit Target dialog box



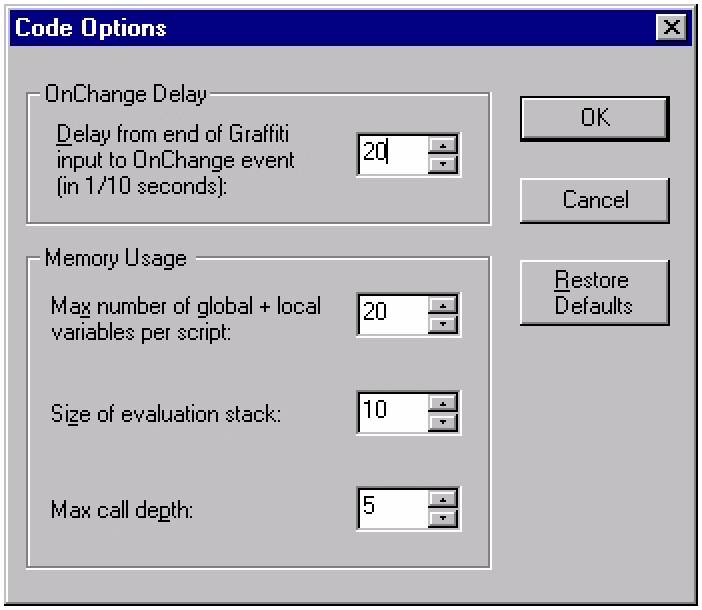
Target name: Palm

Platform: Palm

Copy environment from: Palm

- **Options...:** Displays the **Code Options** dialog box, which allows you to set options that control how MobileApp Designer code works on handheld devices. Click the **Restore Defaults** button to reset all options to the default values. When you are finished editing the code options, click the **OK** button to save all changes or click the **Cancel** button to close without saving your changes. An example Code Options dialog box is shown in the following figure:

Figure 5.23 Code Options dialog box



Code Options

OnChange Delay

Delay from end of Graffiti input to OnChange event (in 1/10 seconds): 20

Memory Usage

Max number of global + local variables per script: 20

Size of evaluation stack: 10

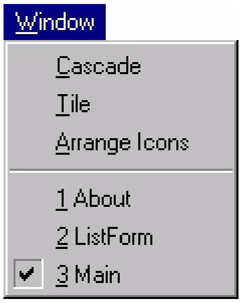
Max call depth: 5

- **OnChangeDelay:** Sets the delay between the end of Graffiti input to the firing of the OnChange event in increments of one-tenth of a second. The default setting is 20, which equals two seconds.
- **Memory Usage:** The three items in this group set memory usage options.
    - **Max Number of global + local variables per script:** Sets the maximum number of global and local variables per script.
    - **Size of Evaluation Stack:** Sets the size of the evaluation stack.
    - **Max Call Depth:** Sets the maximum call depth.

MobileApp Designer Window menu

The **Window** menu allows you to select and arrange the open windows on the MobileApp Designer desktop.

Figure 5.24 MobileApp Designer Window menu



The **Window** menu contains the following options:

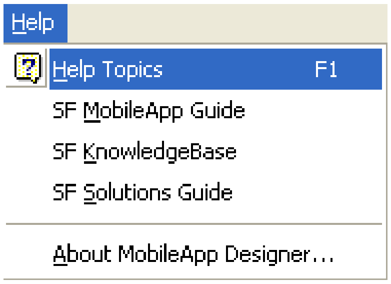
- **Cascade:** Cascades all open windows on the desktop.
- **Tile:** Tiles all open windows on the desktop.

- **Arrange Icons:** Moves the icons of minimized windows to the bottom of desktop.
- **Open windows list:** Click the desired window name to bring it to the front and make it active. The currently active window has a check mark to the left of its name in this list.

MobileApp Designer Help menu

The **Help** menu provides access to online help for and information about MobileApp Designer.

Figure 5.25 MobileApp Designer Help menu



The **Help** menu contains the following options:

- **Help Topics:** Displays the main online help window for MobileApp Designer. For information on using online help, click the **Start** button on the Windows Taskbar and click **Help**.
- **SF MobileApp Guide:** Displays the Satellite Forms MobileApp Guide PDF document in Adobe Acrobat.
- **SF KnowledgeBase:** Opens the Satellite Forms KnowledgeBase, a searchable help file containing known problems and solutions, as well as How-To guides for Satellite Forms. The KnowledgeBase is also available online on the Satellite Forms website.
- **SF Solutions Guide:** Opens the Satellite Forms Solutions Guide, a searchable help file collection of numerous solutions for Satellite Forms developers, organized into various categories. The Solutions Guide is also available online on the Satellite Forms website.
- **About MobileApp Designer...:** Displays the About box, which displays the Satellite Forms MobileApp Designer version and build numbers and copyright notice. If you need to contact technical support, be sure you have the build and version numbers handy.

MobileApp Designer toolbars

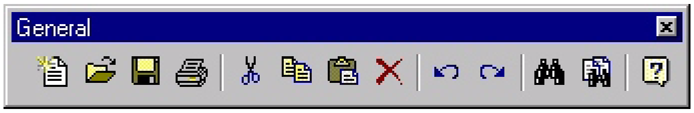
MobileApp Designer includes three dockable toolbars that provide one-click shortcuts to most common tasks. You can drag any of the toolbars off the MobileApp designer window and drop them anywhere on your Windows desktop. Whenever MobileApp Designer is the active application, the undocked toolbars are visible. If you prefer to dock a toolbar, drag it to the desired location just below the menu bar and drop it there. You can arrange the three toolbars in any order, in one or more rows. Note that it is possible that you may not be able to see all of the icons on all of the toolbars in their default layout, depending on your desktop PC screen size. If you can only see some of the icons on the right-most toolbar, for example, drag that toolbar down to the next row, or along the left side of the MobileApp Designer window, so that all of the icons are visible. You should be able to see all of the toolbar icons that are present in the sample images below.

Tip Hover the mouse cursor over a toolbar button to display the name of the button in a tooltip and a description of the what the button does in the status bar.

General toolbar

Several MobileApp Designer File and Edit menu options are available on the General toolbar, as shown in the following figure:

Figure 5.26 MobileApp Designer General toolbar (undocked)



The following table lists and describes the MobileApp Designer General toolbar buttons:

Table 5.9 MobileApp Designer General toolbar buttons

Button	Description
📄	**New Project:** Opens a new Satellite Forms project.
📂	**Open:** Displays a standard file Open dialog box that allows you to open an existing project.
💾	**Save:** Saves the current project. If the current project is new and you have not saved it, MobileApp Designer opens the Save As dialog box, which allows you name the project and select the directory in which to save it.
🖨	**Print:** Displays the standard Print dialog box, which allows you to set printing options and to print the current form.
✂	**Cut**: Cuts the selected item from the project and places it on the clipboard. If you select multiple items, all are cut to the clipboard. You can paste items from the clipboard into the same project or into a different project.
⎘	**Copy**: Copies the selected item and places it on the clipboard. If you select multiple items, all are copied to the clipboard. You can paste items from the clipboard into the same project or into a different project.
📋	**Paste**: Pastes the item from the clipboard into the current project. If the clipboard contains multiple items, all are pasted into the current project. You can paste items from the clipboard into the same project or into a different project.
✕	**Delete:** Deletes the selected item from the project. Some items, such as scripts, allow you to select Undo to restore the most recent deletion. Controls and most other items do not.
↶	**Undo:** Reverses the preceding action, for example, restoring the most recent deletion from a script. This button is available only if the preceding action can be undone.

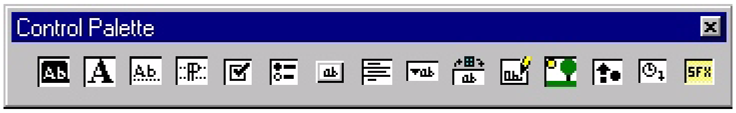
Table 5.9 MobileApp Designer General toolbar buttons *(Continued)*

↷	**Redo:** Repeats the preceding action, for example, restoring the portion of a script removed with the Undo button. This button is available only if the preceding action can be re-done.
	Find: Applies only to scripts. Displays the Find dialog box, which allows you to specify the text to search for in the current script and other search options. See Figure 5.10 on page 81 for an example Find dialog box.
	Find in Project: Applies only to scripts. Displays the Find in Project dialog box, which allows you to specify the text to search for in all scripts in the current project and other search options. See Figure 5.11 on page 81 for an example Find in Project dialog box.
?	**Help:** Displays the main online help window for MobileApp Designer. For information on using online help, click the **Start** button on the Windows Taskbar and click **Help**.

MobileApp Designer Control Palette toolbar

The **Control Palette** toolbar provides access to all of the controls you can place on forms or in applications. Click the desired control button to place that control on the current form, then position the control on the form by dragging it or by setting the Dimensions properties in the Propertyspace palette. An example Control Palette toolbar is shown in the following figure:

Figure 5.27 MobileApp Designer Control Palette toolbar (undocked)



The following table lists and describes the MobileApp Designer Control Palette toolbar buttons:

Table 5.10 MobileApp Designer Control Palette toolbar buttons

Button	Description
Ab	**Title Control:** Each form or page can only have one Title control.
A	**Text Control:** The handheld device cannot change the text a Text control displays. Use Text controls to label other controls.
Ab	**Edit Control:** Displays and optionally edits a specific field from the current record of the form's linked table in a single line.
¶	**Paragraph Control:** Works exactly like an Edit control except that it allows the data to occupy multiple lines.
☑	**Check Box Control:** Displays and optionally edits a True/False field from the current record of the form's linked table: checked equals a field value of True, cleared equals a field value of False.
●	**Radio Button Control:** Works like a Check Box control except that the selection in a group of controls bound to the same table field is mutually exclusive. Radio Button controls can display and edit Numeric fields.

Table 5.10 MobileApp Designer Control Palette toolbar buttons *(Continued)*

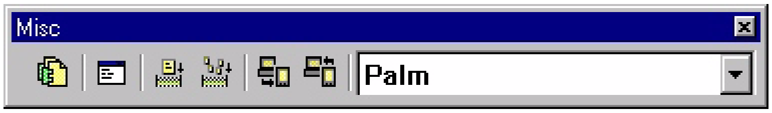
ab	**Button Control:** Performs an action, such as jumping to another form, returning to the previous form, or creating a new record.
☰	**List Box Control:** Displays multiple records from one or more columns of data from the table linked to a form.
▾ab	**Drop List Control:** Displays a list of items from which a user can choose. A Drop List control is the space-saving equivalent of a Lookup control.
ab	**Lookup Control:** Displays a list of items from which a user can choose. Use Lookup controls to display information in a user-friendly format by using a linked lookup table to retrieve information from another table that contains the user-readable data.
ab	**Ink Control:** Allows you to collect signatures or sketches from the handheld device. The drawing or signature created in the Ink control is compressed and saved in the Binary field designated as the control's data source.
🖼	**Bitmap Control:** Places a bitmap on a form. Use this control to add pictures, logos, and other interesting visual touches to your forms.
⇧	**Graffiti Shift Indicator Control:** Places an indicator graphic on the handheld device's screen that shows the shift status of the Graffiti handwriting recognizer. This control displays different symbols for lowercase, shifted, or caps lock Graffiti mode. It is not available for Pocket PC applications.
⏲	**Auto Stamp Control:** Automatically enters a date or time stamp in a field. Display the time/date stamp by adding an Edit control linked to the same data source as the Auto Stamp control.
SFX	**SFX Custom Control:** Inserts an SFX Custom control in a form. This button is only active if an SFX Extension has been added to the current project.

For more information on using controls and setting control properties, see Using controls on page 124.

MobileApp Designer Misc toolbar

Several MobileApp Designer View, Handheld, and Build menu options are available on the Misc toolbar, as shown in the following figure:

Figure 5.28 MobileApp Designer Misc toolbar



The following table lists and describes the MobileApp Designer Misc toolbar buttons:

Table 5.11 MobileApp Designer Misc toolbar buttons

Button	Description
	Manage Extensions: Displays the **Available Extensions** dialog box, as shown in Figure 5.15 on page 86, for an example and more information.
	Show Scripts: Displays the **Application Scripts** window on the desktop. See Chapter , Satellite Forms Scripting Language Reference, on page 257 for more information.
	Compile Script: Compiles the open script. This button is available only when a script is open in the Application Scripts window.
	Rebuild All: Rebuilds the current project for the selected application target. The selected application target is shown in the Select Target drop-list.
	Download App & Tables: Downloads the current application and its tables to a handheld device.
	Upload Tables: Uploads an application’s tables from the handheld device to the development computer where MobileApp Designer can access them.
Palm	**Select Target:** Sets the application build target for the project. A project can contain multiple application targets, each of which can have its own forms, controls, properties, and other properties.

Chapter 6
Creating your Application

This chapter lists and describes steps required for developing your Satellite Forms application. It is organized into the following sections:

- Planning your application: Advice about the planning stage of a Satellite Forms application.
- Overview: phases of application development: A brief description of the three phases of Satellite Forms application development.
- Creating and editing forms: Instructions for creating forms using MobileApp Designer.
- Using controls: Definitions and descriptions of all available controls.

This chapter assumes you are familiar with the MobileApp Designer application. Refer to Overview, on page 39 and MobileApp Designer Reference, on page 67 for a review if needed. Many of the concepts involved in creating an application are also illustrated in detail in Sample Application: Work Order, on page 527.

Planning your application

A Satellite Forms application is a collection of tables and forms designed to allow users to browse existing information, collect new information, or both.

It is important to think through a proposed application before starting a project. You should clearly define the purpose and the information requirements of the application before you start working with MobileApp Designer to create the tables and forms. If you complete the planning task before you create forms and tables, you will do less backtracking and redesigning and the completed application will function in a manner that facilitates, complements, and enhances the end-users' performance.

Elements of good planning:

A well planned application typically addresses each of the following elements:

- End user requirements: who will use your application, and for what purpose?
- Application requirements: what does the application need to do?
- Application design: how will the application satisfy the requirements?
- Application development: how will the application be implemented?
- Application testing: does the application work as planned?

- Deployment and maintenance: how will the application be delivered to end users and maintained after deployment?

You should always clearly define the data to be accessed by end users, the information to be collected and merged into the database, and the forms to be built with MobileApp Designer. Careful planning makes it possible to design compact, concise forms that make the most of the handheld device's screen size. Although it is possible to create forms that are many pages long, in general a short, concise form will result in a more effective application.

You must also decide how data should move between handheld devices and your database. If you divide your application into several efficient pieces, information moves more smoothly. For example, on the desktop, Microsoft Outlook is one application, similar to your desktop application or database. On the handheld, several smaller applications replace Outlook, for example, the Palm Address Book, Calendar, Memo Pad, To Do List, and Mail applications. Your Satellite Forms application should be similar to these smaller applications, rather than one large application that tries to do too much.

The four major questions to ask when planning an application:

When you are planning your Satellite Forms application, ask the following questions:

1 What kinds of information do end users need to access?

Sales staff would need customer account information, which may include a customer's location and contact information, inventory, scheduled visits, and so on.

2 What kinds of information do end users need to update or add?

Using the sales example again, this type of data might include new orders, returns, updating inventory numbers, and so on.

3 What format is required to store the data in questions 1 and 2?

This question relates to the fields that appear in the application's tables. These tables contain the information described by the answers to the first two questions. How many fields are required? What is the best way to group these fields? What data types are needed to display or record this data?

4 What database format should you use?

Satellite Forms supports both .DBF (dBase V) and .MDB (Access) database formats to store its transfer tables. In your project, use the Project Properties dialog box, as shown in Figure 5.13 on page 83, to set the desktop database format to use for each application target.

- If you are planning to use Access 2000 or later for your Palm OS applications, you must specify .MDB as the desktop database format.

After you have planned your Satellite Forms application, you are ready to start creating the application using MobileApp Designer.

Overview: phases of application development

This section summarizes the complete process of creating an application, integrating the application with your database, and deploying the application to your end users. Additional examples and instructions appear in the sample application described in

Sample Application: Work Order, on page 527. If needed during development, a complete reference for working with MobileApp Designer is provided in MobileApp Designer Reference, on page 67.

Developing a typical Satellite Forms application can be broken into three phases: Phase 1 – Working with MobileApp Designer, Phase 2 – Integrating applications with your database management system, and Phase 3 – Deploying your application. A summary of each of the three phases is provided below, followed by details on implementing Phase 1. Phase 2 is covered in Integrating a Satellite Forms database with a Corporate database, on page 196. Phase 3 is covered in Deploying your Application, on page 229.

Phase 1 – Working with MobileApp Designer

In Phase 1, you use MobileApp Designer to create, configure, and test Satellite Forms applications for handheld devices. This process consists of ten steps after a new project has been created:

1. Create tables by adding and configuring columns (fields) and entering sample data using the table editor. These tables hold the data to be displayed, edited, or both, on the handheld device.
2. Create forms and associate the forms as appropriate with the tables containing the data to be displayed or modified. These forms are the screens or pages that users see when the application is downloaded and run on the handheld device.
3. Add controls to forms, set their properties, associate as appropriate with a specific table column, and position them on the form.
4. Add scripts to your application to enhance its functionality. For more information, see Satellite Forms Scripting Language Reference, on page 257.
5. Add control actions, table filters, extensions, and color to your application to enhance its functionality. For more information, see Using Actions, Filters, Extensions, and Color, on page 177
6. Configure your application's properties, including the application's name, the initial form, and other options.
7. Create an icon image for your application.
8. Download the application and its tables to the handheld device to check the appearance and operation of your forms and controls using the sample data you input in Step 1. For instructions on downloading an application to a handheld, see Installing the engine and downloading the application on page 175.
9. Upload application tables back into MobileApp Designer to make sure any updates or new data entries made using the handheld device were transferred correctly to the associated tables in the application. For instructions on uploading application tables from the handheld device to MobileApp Designer, see Installing the engine and downloading the application on page 175.

Details on creating your application are provided later in this chapter. After you have created your application, you are ready to integrate the application with your database management system in Phase 2.

Phase 2 – Integrating applications with your database management system

Once you have completed Phase 1, you have an operational handheld device application. Phase 2 focuses on integrating the application with your database management system. To accomplish this integration, you must complete the following steps:

1. Link the database tables created by Satellite Forms with the database application.
2. Write code to extract data from the database and copy it into the Satellite Forms tables. This code prepares a table or tables to be downloaded to the handheld device.
3. Write code to transfer data from the associated Satellite Forms database tables on the handheld device back to the database and merge as desired.
4. Write code to handle sync events, that is, to transfer information to and from the handheld device. This code calls the code in Steps 2 and 3 as required.

Detailed information and instructions for completing Phase 2 appear in Integrating a Satellite Forms database with a Corporate database, on page 196.

Phase 3 – Deploying your application

After you have completed Phase 2, you have an operational handheld device application that is integrated with your database management system. Phase 3 focuses on distributing your application to your end users. To deploy your application, you must complete the following steps:

1. Modify the HotSyncStatus or ActiveSync handler of your PC application.
2. Load the Satellite Forms RDK redistributable components to any computers that will be used to install or synchronize with Satellite Forms applications onto handheld devices.

Note You can create install disks from the provided disk images or write your own installation program to accomplish this step.

3. Load all files associated with your application onto the PC.
4. Load all files associated with your Satellite Forms application onto any computers that will be used to install or sync with Satellite Forms applications onto handheld devices.
5. Load the necessary files onto handheld devices.
6. Register the Satellite Forms conduit with ActiveSync or HotSync Manager and restart it.
7. Install the Satellite Forms engine and other files, if necessary, onto the handheld devices.
8. Install the Satellite Forms application (forms and tables) onto handheld devices.

You can use the Satellite Forms RDK Install Utility (RDKINST.EXE for Palm OS and CeRdkInst.EXE for Pocket PC) to simplify the loading of the necessary files onto handheld devices. Detailed information and instructions for completing Phase 3 appear in Deploying your Application, on page 229.

Phase 1: Working with MobileApp Designer

This section contains detailed instructions and information about Phase 1 of developing your Satellite Forms application. It assumes that you know how to open a new project and perform basic functions with MobileApp Designer. Refer to Overview, on page 39 or MobileApp Designer Reference, on page 67 if needed during development.

You will complete the following steps in Phase 1:

1. Work with tables.
2. Create and edit forms.
3. Create menus you want to appear in your application.
4. Use controls to add functionality to your application.
5. Enhance your application's functionality with actions, filters, extensions, and SFX plug-ins and custom controls.
6. Configure your application's properties.
7. Download the application to the handheld device from MobileApp Designer.
8. Upload application tables back into MobileApp Designer from the handheld device.

Working with tables

Most applications contain tables. This section explains how to create and edit or import tables for use when developing applications with MobileApp Designer. Like other relational databases, Satellite Forms uses tables to store and retrieve information. An application created with MobileApp Designer can use one or many tables. Tables in Satellite Forms have the following requirements:

- Each table in a Satellite Forms application must have a unique name.
- Each table has rows and columns – that is, records and fields.
- Each field is labeled with a field name describing the information stored in the field (for example, CLIENTNAME).
- Each field is assigned a specific data type that determines the characters and format the field accepts. For example, only numbers can be stored in a numeric field.

Satellite Forms tables support the following field data types:

- **Character:** Accepts any printable character: letters, numbers, and symbols.
- **Numeric:** Accepts numbers only, such as prices, inventory quantities, ID numbers, and so on. Note that phone numbers, zip codes, and numbers

representing dates and times should not be entered into numeric fields. Use the Character data type for phone numbers and zip codes. Dates and times have their own corresponding data types.

- **True/False:** Accepts only T for true or F for false. Use for Boolean data.
- **Date:** Accepts only date information in MM/DD/YYYY format in MobileApp Designer. On the handheld device, dates are displayed and entered in a country–specific format according to the **Preferences** settings on the handheld device.
- **Time:** Accepts only time information in the HH:MM am/pm format in MobileApp Designer. On the handheld, times are displayed and entered in a country–specific format according to the **Preferences** settings on the handheld device.
- **Binary:** Accepts input from an Ink control for signatures, drawings, and so on. This data appears on the desktop computer as an OLE object.
- **Time Stamp:** Accepts an eight-bit integer used to establish compatibility with imported Oracle Lite tables.

Creating a new table

To create a new table in the current project, select **Edit > Insert Table** from the MobileApp Designer menu. This opens the **Table** dialog box with a default table and column, as shown in the following figure.

Figure 6.1 Table dialog box

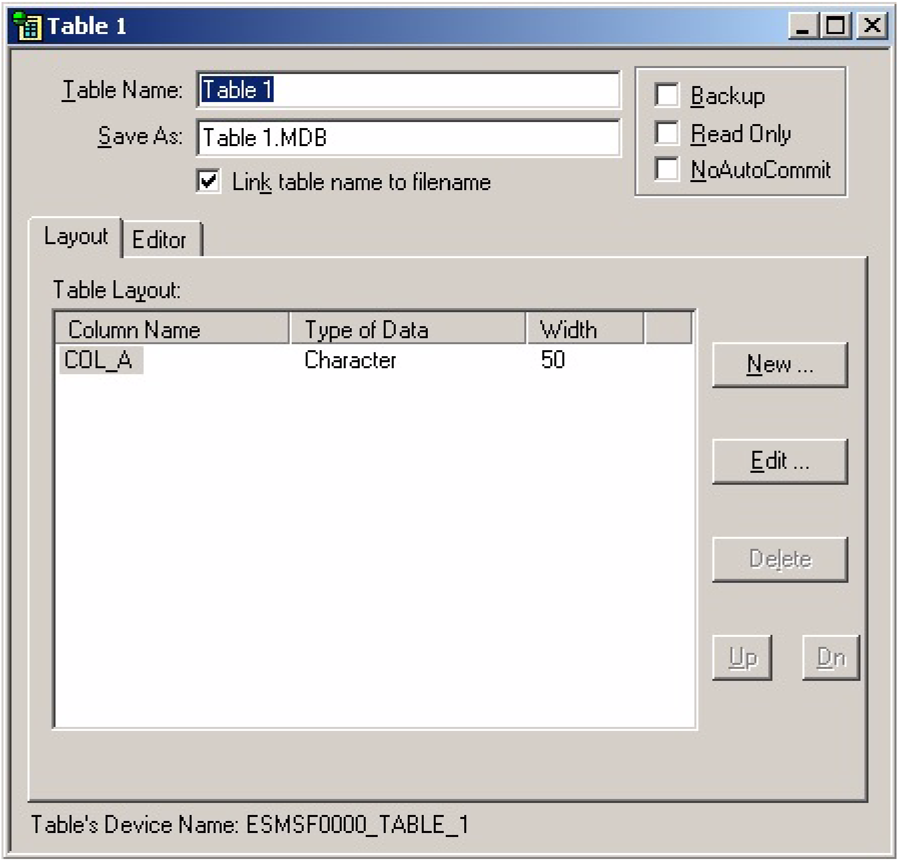


Table Name: Table 1

Save As: Table 1.MDB

☑ Link table name to filename

☐ Backup

☐ Read Only

☐ NoAutoCommit

Table Layout:

Column Name	Type of Data	Width
COL_A	Character	50

Table's Device Name: ESMSF0000_TABLE_1

Setting the table name and database options

Type a name for this table into the **Table Name** edit box. This is the name by which all objects in the current project reference the table. Check the **Link table name to filename** box to have MobileApp Designer use the table name to generate the default name of the database file created when you save the project. Change the database file name by typing a new name in the **Save As** edit box, if desired.

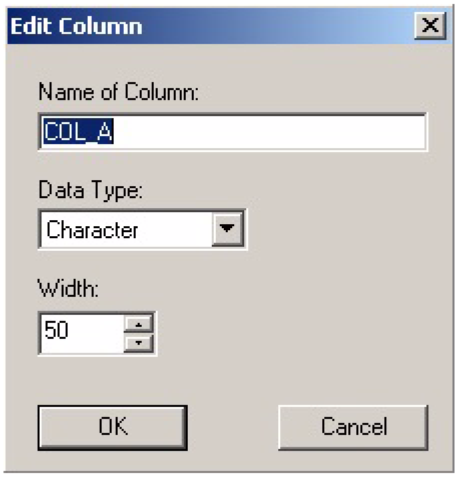
The **Backup**, **Read Only**, and **NoAutoCommit** options allow you to set some special behaviours, or attributes, for each table:

- The **Backup** option applies to the Palm OS platform only (it has no effect on the Pocket PC platform whether or not it is checked) and is used to specify whether Palm HotSync should make an extra backup copy of the table during the HotSync process.
- The **Read Only** option allows you to make the table so that it cannot be modified. The records in the table can be viewed but they cannot be changed. This option applies to both Palm OS and Pocket PC platforms. On the Pocket PC platform, there is a performance advantage to using read only tables, as they allow your application to close faster than standard read/write tables. Therefore, if your application design is such that this particular table cannot and should not be modified on the handheld, it makes sense to ensure the Read Only option is checked.
- The **NoAutoCommit** option applies to the Pocket PC platform only (it has no effect on the Palm OS platform whether it is checked or not). This option defines a table that behaves in some ways like a regular read/write table, and in some ways like a read only table, and is generally considered an advanced option. The NoAutoCommit option means that the table can be modified like a regular read/write table, but that none of the changes to the table are saved automatically. In order to save changes to a NoAutoCommit table, your application has to save or "commit" those changes in script by calling the Tables("tablename").CommitData method. The reasons why you might want to use this table option are explored in a Satellite Forms KnowledgeBase article titled "How To Make PocketPC PDB Apps Close Faster" which you can find in your locally installed KnowledgeBase or online at http://www.satelliteforms.net/knowledgebase.htm.

Setting the table layout

When you create a new table, the **Layout** tab on the **Table** dialog box contains a single, default column. Click the column name and then click the **Edit...** button or double-click the column name to display the **Edit Column** dialog box, as shown in the following figure:

Figure 6.2 Edit Column dialog box



Edit Column

Name of Column: COL_A

Data Type: Character

Width: 50

Type a name for the column. Column names can be a maximum of ten characters which must be the letters A-Z, the numbers 0-9, or the underscore character(_). Column names should reflect the contents of the field being created and must be unique within the same table. All objects in the current project reference the column by this name.

Column data types

Columns in Satellite Forms tables support the following data types:

- **Character**: Any printable character: letters, numbers, and symbols.
- **Numeric**: Numeric data: prices, inventory quantities, and so on.
- **True/False**: A standard Boolean data type. Fields of this type accept either T for true or F for false. True/False fields are frequently linked to Check Box controls. See Check Box control on page 136 for more information.
- **Date**: Dates in MM/DD/YYYY format, for example, 11/21/2003 or 5/6/2004 in MobileApp Designer only. On the handheld device, dates are displayed and entered in a country–specific format according to the settings in the handheld device’s preferences panel.
- **Time**: Times in HH:MM AM/PM format (10:42 PM, 7:11 Am) in MobileApp Designer only. On the handheld device, times are displayed and entered in a country–specific format according to the settings in the handheld device’s preferences panel.
- **Binary**: Ink control data format for signatures, drawings, and so on. This data appears on the desktop computer as an OLE object after uploading data from the handheld device.
- **Time Stamp**: An eight-bit integer used to maintain compatibility with imported Oracle Lite tables.

The **Character** and **Numeric** data types require a column width setting. The column width determines the maximum number of characters a field accepts. The column width need not be the same as the width of the control that displays the information on the handheld device.

- The Character data type column width must be in the range 1 to 32,767.

- The Numeric data type column width must be in the range 1 to 19.

Tip To conserve space and memory, set the column widths to the smallest useful size possible for each column.

The **Numeric** data type supports decimal places from 0 to 6 or floating point.

Date and time shortcuts

To make the entry of dates and times on the handheld device easier, you can take advantage of a number of shortcuts Satellite Forms provides.

Normally when you enter dates, you use the format specified in the handheld device's preferences. Satellite Forms applications default to the current month, year, and century allowing you to omit these items in date entries, as appropriate.

For example, assume today is June 30, 2003 and the preferred date format on the handheld device is MM/DD/YY. The following table illustrates the date value Satellite Forms records for the given date value entered:

Table 6.1 Satellite Forms date format example

Value Entered	Value Recorded
5	6/5/2003
5/3	5/3/2003
5/3/80	5/3/1980
5/3/2005	5/3/2005

Likewise, you normally use the format HH:MM or HH:MM:SS to enter times. Satellite Forms assumes all times are AM unless entered as PM. You can shorten AM and PM by entering an `a` or a `p` instead. In addition, Satellite Forms applications default to zero all missing time elements, so you can omit the seconds or minutes portion of a time as appropriate. The following table illustrates the time value Satellite Forms records for the given time value entered:

Table 6.2 Satellite Forms time format example

Value Entered	Value Recorded
1	1 AM
1a	1 AM
1:15 a	1:15 AM
1:15 am	1:15 AM
1p	1 PM
1:15 p	1:15 PM
1:15 pm	1:15 PM

Adding/deleting columns

To add a new column, click the **New...** button or double-click in the white space below the existing columns to display the **Create New Column** dialog box. This dialog box works exactly like the **Edit Column** dialog box, as shown in Figure 6.2 on page 106. The default name for the new column is COL_*X*, where *X* is the next available letter. Type a new name and set up the column's data type and options. Click the **OK** button to add the new column to the table. Repeat this process as needed to add all necessary columns to the table.

To delete an existing column, click the desired column and then click the **Delete** button. MobileApp Designer deletes the column immediately, without prompting you first.

Setting column order

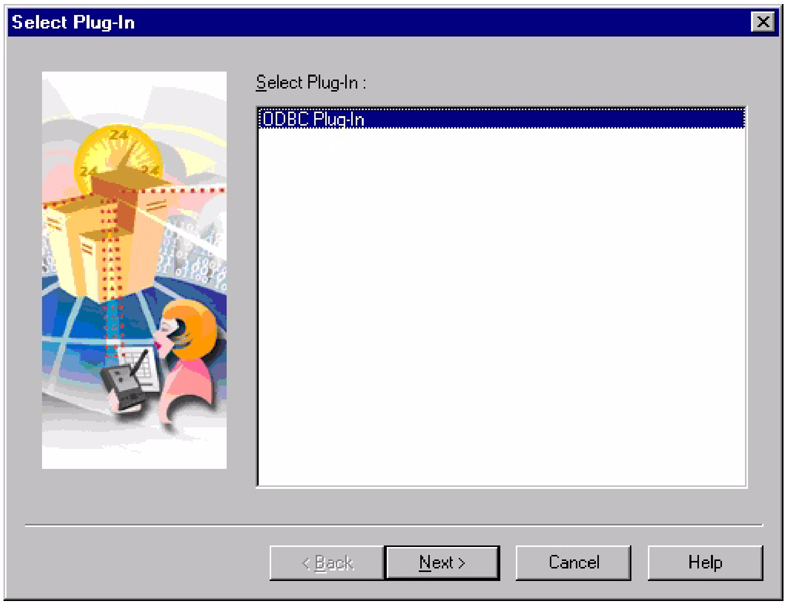
To change the order of the columns in a table, click the column you would like to move and then click the **Up** or **Dn** button to position the column as desired. Repeat this process until the columns are in the desired order.

Importing tables

MobileApp Designer allows you to import a database schema from an ODBC data source. The structure of these tables, but not the data, is automatically created in the current project. This feature saves you from having to create tables manually in MobileApp Designer when they already exist in an ODBC data source.

Select **Edit > Import Table...** from the MobileApp Designer menu to start the **Import Table** wizard, as shown in the following figure:

Figure 6.3 Import Table wizard, Select Plug-In step


Select Plug-In

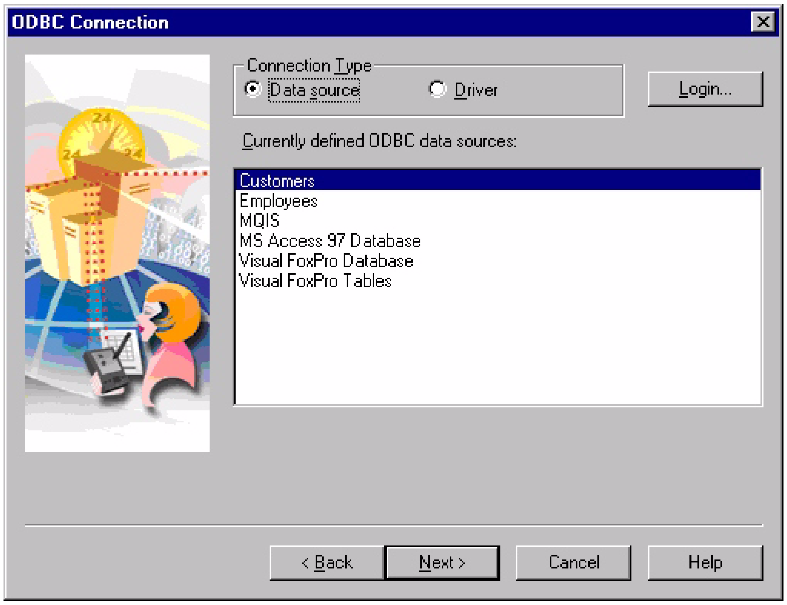
Select Plug-In :

ODBC Plug-In

Select the desired plug-in and click the **Next >** button.

The next step allows you to select the desired connection type and then a data source based on the connection type, as shown in the following figure:

Figure 6.4 Import Table wizard, ODBC Connection step, Data source selected



Select either **Data source** or **Driver** as the connection type. Depending on the option you select, the next wizard step displays all the data sources or drivers present on your development computer. The steps required to import a table from a data source and from a driver are explained in the following sections. See Importing tables using a driver connection on page 109 for instructions on using a driver connection.

Importing tables using a data source connection

When you select **Data source** as the connection type, the Import Table wizard lists available ODBC data sources. If the data source you want to use requires a login, click the **Login...** button and type your user name and password. Select the desired ODBC data source in the list and click the **Next >** button.

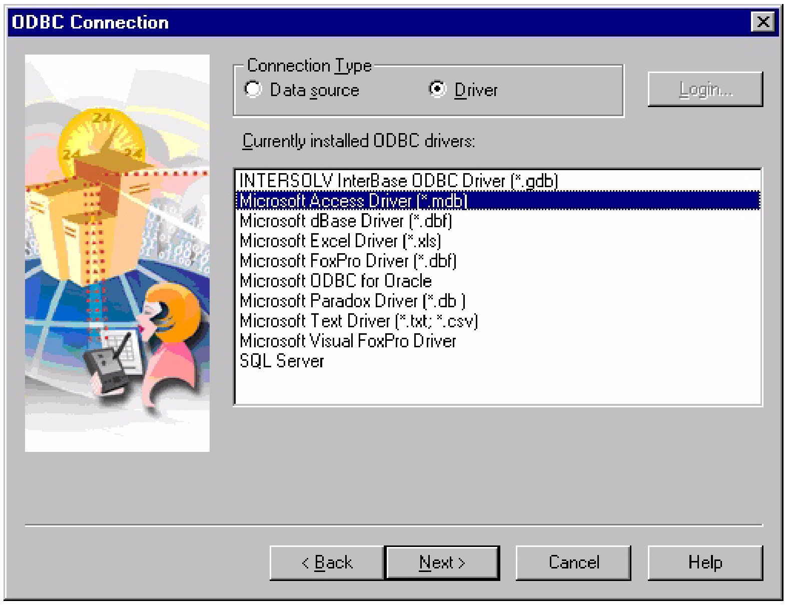
Tip If the data source you want to use is not listed, use the Windows ODBC Data Sources Control Panel applet to define the desired data source. See Windows Help for more information.

At this point, the process of importing a table schema from data source or a driver is identical. See Importing the table schema on page 110 for instructions.

Importing tables using a driver connection

When you select **Driver** as the connection type, the Import Table wizard lists the ODBC drivers installed on your development computer as shown in the following figure:

Figure 6.5 Import Table wizard, ODBC Connection step, Driver selected



Connection Type
○ Data source ● Driver

Currently installed ODBC drivers:

- INTERSOLV InterBase ODBC Driver (*.gdb)
- Microsoft Access Driver (*.mdb)
- Microsoft dBase Driver (*.dbf)
- Microsoft Excel Driver (*.xls)
- Microsoft FoxPro Driver (*.dbf)
- Microsoft ODBC for Oracle
- Microsoft Paradox Driver (*.db)
- Microsoft Text Driver (*.txt; *.csv)
- Microsoft Visual FoxPro Driver
- SQL Server

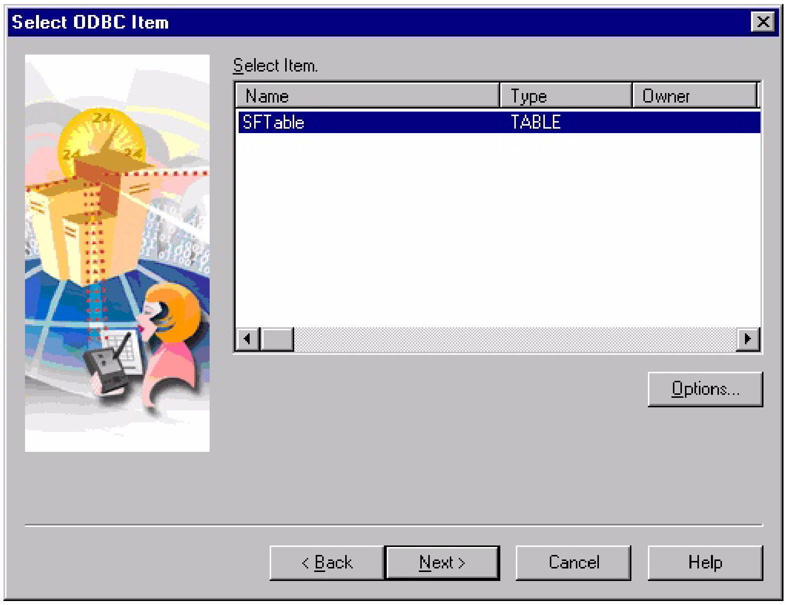
Select the desired ODBC driver in the list and click the **Next** button. Use the database driver to select the desired database or instance.

Note When you click the **Next >** button, the wizard accesses the selected database driver. Since every driver is different, instructions for selecting a database file or instance cannot be provided here. See the online help or user manuals supplied with the database driver for more information.

Importing the table schema

After you have selected the desired data source or driver, the next step is to select the table whose schema you want to import, as shown in the following figure:

Figure 6.6 Import Table wizard, Select ODBC Item step



Select Item.

Name	Type	Owner
SFTable	TABLE	

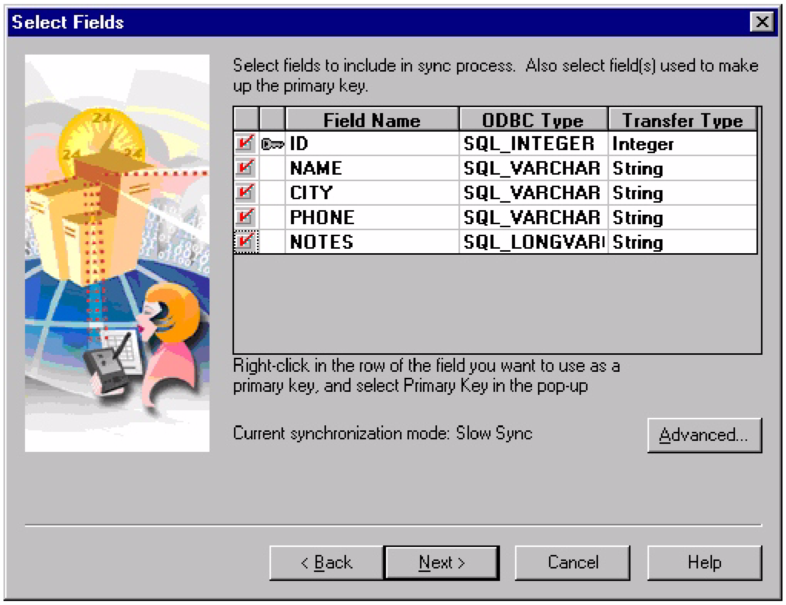
Select the desired table in the list and click the **Next >** button.

Tip To display additional items stored in the selected ODBC data source, including views, system tables, and aliases, click the **Options...** button and check the desired options.

Check the box to the left of each field you want to import. To designate a field as a primary key, right-click the desired field and select **Primary Key** from the popup menu. Repeat this process for additional primary key fields. This option is available only for database types that support primary keys.

Tip Click the **Advanced...** button to set synchronization options and a SQL SELECT statement WHERE clause, if desired.

Figure 6.7 Import Table wizard, Select Fields step



Select Fields

Select fields to include in sync process. Also select field(s) used to make up the primary key.

		Field Name	ODBC Type	Transfer Type
☑	🔑	ID	SQL_INTEGER	Integer
☑		NAME	SQL_VARCHAR	String
☑		CITY	SQL_VARCHAR	String
☑		PHONE	SQL_VARCHAR	String
☑		NOTES	SQL_LONGVARI	String

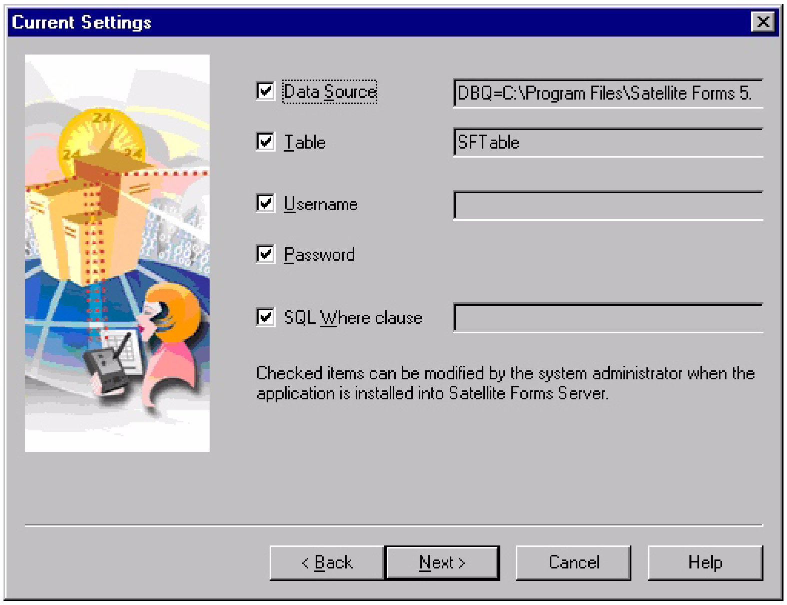
Right-click in the row of the field you want to use as a primary key, and select Primary Key in the pop-up

Current synchronization mode: Slow Sync

When you are finished selecting the fields to be imported and setting the primary keys, click the **Next >** button.

The next wizard step shows you the selected settings for the table to be imported as shown in the following figure:

Figure 6.8 Import Table wizard, Current Settings step



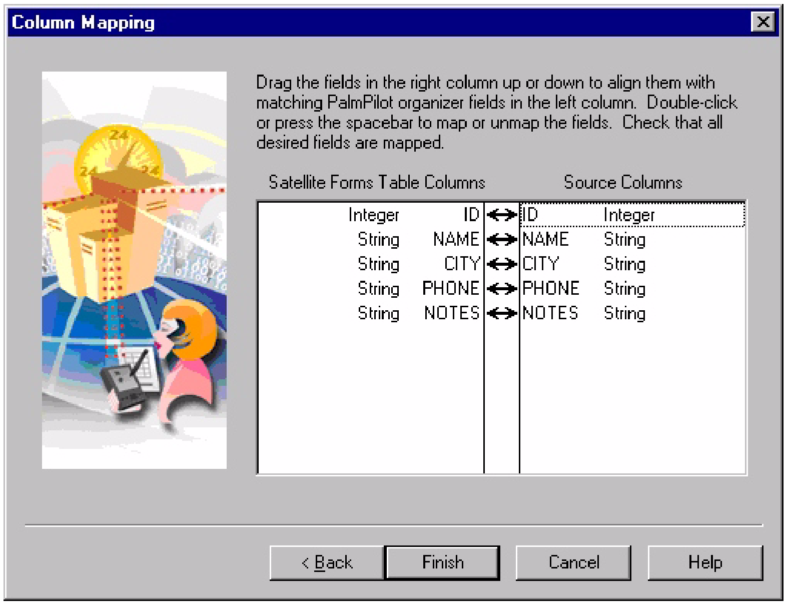
- ☑ Data Source: DBQ=C:\Program Files\Satellite Forms 5.
- ☑ Table: SFTable
- ☑ Username:
- ☑ Password
- ☑ SQL Where clause:

Checked items can be modified by the system administrator when the application is installed into Satellite Forms Server.

Clear the check box to the left of any item that you do not want to allow the system administrator to modify when the application is installed on the server. When you are finished setting administration option, click the **Next** >button.

The final wizard step shows you the default column mapping for the table to be imported as shown in the following figure:

Figure 6.9 Import Table wizard, Column Mapping step



Drag the fields in the right column up or down to align them with matching PalmPilot organizer fields in the left column. Double-click or press the spacebar to map or unmap the fields. Check that all desired fields are mapped.

Satellite Forms Table Columns			Source Columns	
Integer	ID	↔	ID	Integer
String	NAME	↔	NAME	String
String	CITY	↔	CITY	String
String	PHONE	↔	PHONE	String
String	NOTES	↔	NOTES	String

If any columns are not properly mapped to their Satellite Forms equivalents, click and drag the desired column in the right-hand column until it is aligned with the correct Satellite Forms column and drop it there. When you are finished setting the column mapping click the **Finish** button to import the selected table schema.

Editing table data

After creating or importing a table, you can add or edit data as needed using MobileApp Designer to support your development effort.

Note For instructions on changing the table layout, see Creating a new table on page 104.

The Editor page is not intended for actual data entry into a Satellite Forms application, aside from smaller databases such as droplist items. Your DBMS or sync server application is much faster and more efficient for data entry. Use the Editor page to create a few records for the application so you can test it completely on the handheld device before moving to the integration phase.

To edit table data, click the **Editor** tab. Click the **Insert Rows** button to insert a row above the current row. Click the **Delete Rows...** button to delete the current row. MobileApp Designer prompts you to delete the selected row. An example Editor tab is shown in the following figure:

Figure 6.10 Table dialog box, Editor tab

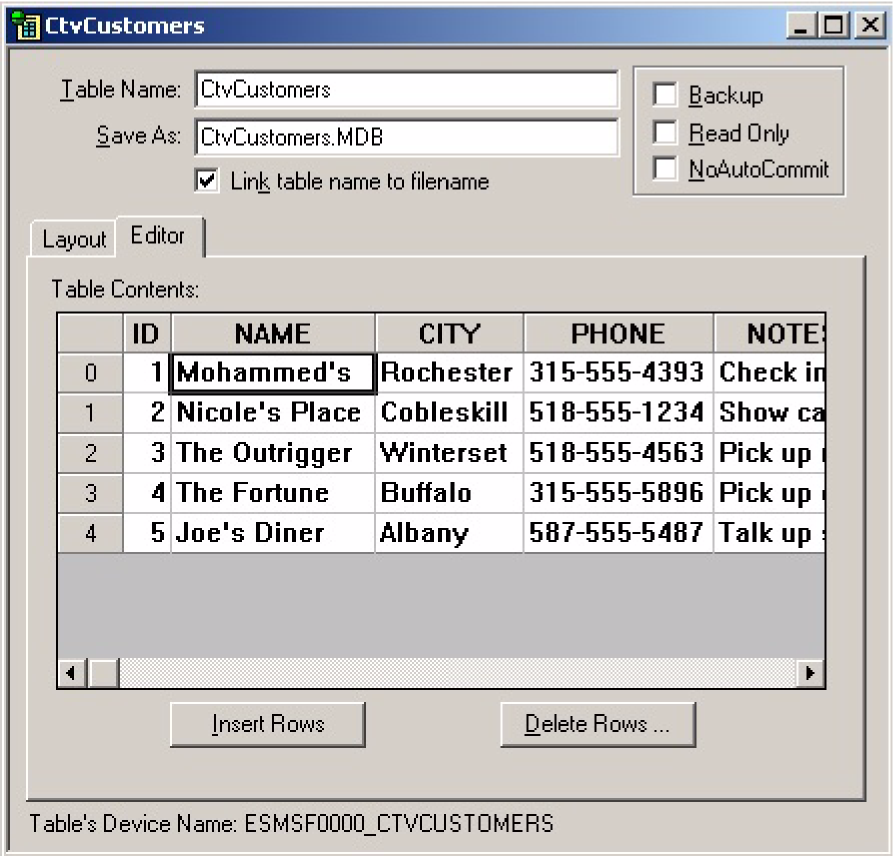


Table Name: CtvCustomers

Save As: CtvCustomers.MDB

☑ Link table name to filename

□ Backup

□ Read Only

□ NoAutoCommit

Table Contents:

	ID	NAME	CITY	PHONE	NOTE
0	1	Mohammed's	Rochester	315-555-4393	Check in
1	2	Nicole's Place	Cobleskill	518-555-1234	Show ca
2	3	The Outrigger	Winterset	518-555-4563	Pick up
3	4	The Fortune	Buffalo	315-555-5896	Pick up
4	5	Joe's Diner	Albany	587-555-5487	Talk up

Table's Device Name: ESMSF0000_CTVCUSTOMERS

To place a Tab character into a cell, type `\t`; to insert a carriage return/line feed into a cell, type `\n`.

Tip When you download the table to a handheld device, these sequences are converted to their proper values.

You can also copy & paste data into the table editor instead of typing it in. To do this, you must first create the empty rows in the table to receive the pasted data, either by using the Insert Rows button, or just by holding down the down cursor button to create new rows. The table you are copying from must have the same column layout, of course. This approach works well when copying some data from a spreadsheet program like Microsoft Excel®, for example.

Creating and editing forms

Forms provide the graphical user interface (GUI) for your applications. Forms display information from a linked table, allow users to add or edit data, and provide other tools that make your handheld applications both useful and easy to use.

Forms can have one or more pages, each page filling the handheld device’s screen. A handheld application can have as many forms as needed to perform its tasks.

Tip Efficient, effective applications have as few forms and as few pages per form as possible.

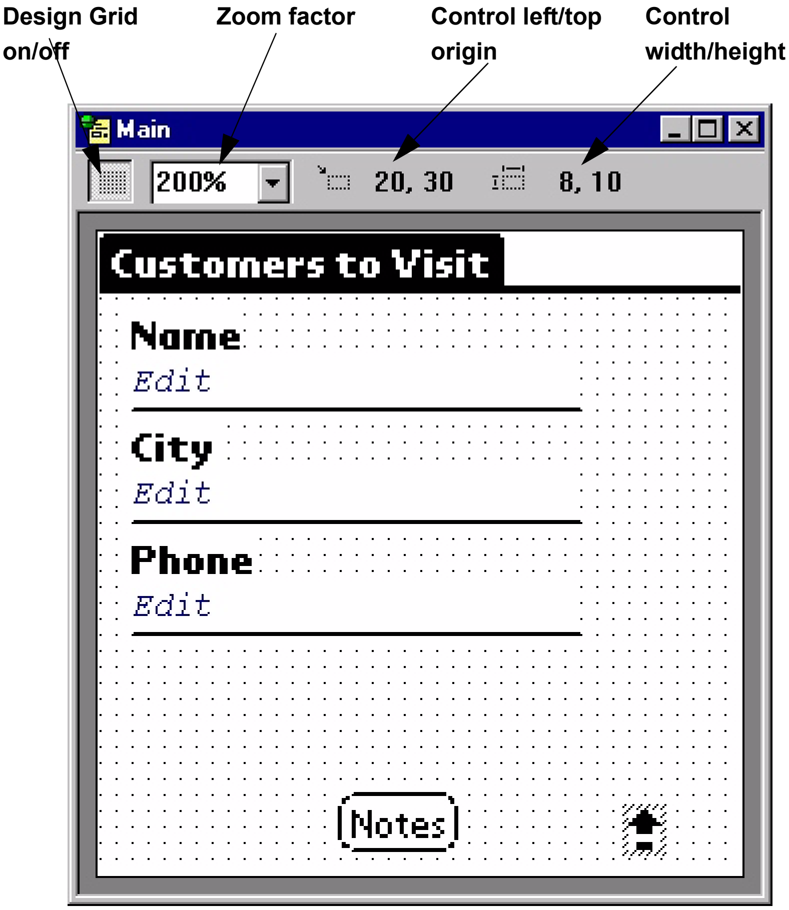
When you open a new project, MobileApp Designer places a blank form on the desktop for you to begin working with.

To add a form to a project, select **Edit > Insert Form** from the MobileApp Designer menu or right-click the **Forms** root icon in the Workspace palette and select **Insert Form** from the context menu. To delete a form from a project, right-click the form’s icon in the Workspace palette and select **Delete Form** from the context menu.

Form design window

The Form design window allows you to turn the design grid on and off and set the zoom factor at which the form is displayed. It also displays the current position and size of the selected control on the form.

Figure 6.11 Form design window



Form design grid

Click the **Grid on/off** button on the Form design window to turn the design grid on or off. When the grid is turned on, as shown in Figure 6.11, control edges snap to the grid points, making it easy to position, size, and line up controls on a form.

The design grid size is five pixels, which limits the position and size of a control to five-pixel increments. Turn off the design grid to gain finer control over the size and position of controls on a form.

To position and line up controls without the design grid, use the keyboard arrow keys to move the selected control one pixel in any direction. To size a control, click **Ctrl** + the desired arrow key. To position, size, and line up controls with greater precision, enter the desired values into the Propertyspace palette for each control on a form.

Form zoom factor

MobileApp Designer displays all Palm OS forms at 200% of actual size by default (for the Pocket PC platform the default zoom level is 100%). Click the arrow to the right of the Zoom factor and select the desired percentage from the list.

Control origin and size indicators

The control origin is the x,y location, in pixels, of the control's upper left corner on the form. The control size is its width and height in pixels. Whenever you select a control on a form, these coordinates appear on the form's toolbar, as shown in Figure 6.11 on page 115.

Form properties

Each form has a series properties that determine its base behavior. These properties set the form’s name, length, associated table, and permissions.

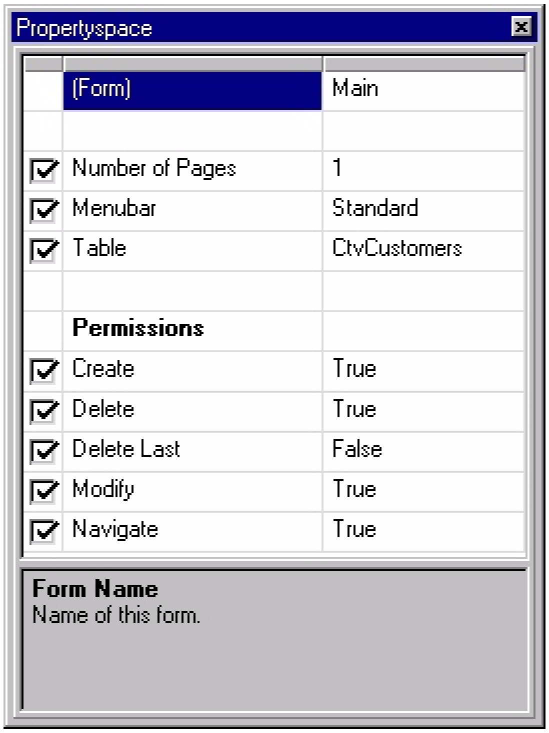
Propertyspace palette

To set form properties, click the open area of the desired form. Make sure you do not click a control on the form. The form’s properties appear in the Propertyspace palette. The Propertyspace palette displays the properties of the selected form or control in a dockable pane.

The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. The left-hand column only appears if the current project has more than one application target defined. An example of a property value that would not be shared between application targets is the position of a control on a form. The position would be different for Palm and Pocket PC application targets because of the different screen sizes of the two platforms. If a property has the target check box checked, and you change that property, that setting is changed for all targets. If you want to make a change to this target only, leaving any other targets unchanged, uncheck the target column and then change the property setting. If you want to revert back to the default setting across all targets, re-check the target column for that property.

The center column, or left-hand column if the current project has just one application target, displays the property. The right-hand column displays the setting for each property. Some settings provide combo boxes from which you can select a valid value.

Figure 6.12 Propertyspace palette (undocked) showing Form properties



	(Form)	Main
☑	Number of Pages	1
☑	Menubar	Standard
☑	Table	CtvCustomers
	Permissions	
☑	Create	True
☑	Delete	True
☑	Delete Last	False
☑	Modify	True
☑	Navigate	True

Form Name
Name of this form.

To dock the Propertyspace palette, drag the window to the desired edge of the MobileApp Designer main window and drop it there. To undock the Propertyspace palette, drag the top of the palette to the center of the desktop and drop it there. The Propertyspace palette works like an ordinary window on the desktop.

Form properties

The properties available for a form are listed in the following table:

Table 6.3 Form properties

Property	**Value**
(Form)	The name of the form. Other forms in the current project refer to the form by this name. Type the desired name.
Number of Pages	The number of pages defined for the current form. Each page is linked to the form's table. Use the spin buttons to set the desired number of pages.
Menubar	Select the name of the Menubar for this form. You cannot select this value before adding a menu to the project.
Table	Select the name of the table to link to this form. You cannot select this value before adding or importing a table into the project.
Permissions: Create	• **True:** Slow users to add new records to the linked table. • **False:** Prevent users from adding new records to the linked table.
Permissions: Delete	• **True:** Allow users to delete records from the linked table. • **False:** Prevent users from deleting records from the linked table.
Permissions: Delete Last	• **True:** Allow users to delete the last (only) record in the linked table, leaving the table empty. • **False:** Clears the last record, but does not delete the last (only) record from the linked table.
Permissions: Modify	• **True:** Allow users to modify existing records in the linked table. • **False:** Prevent users from modifying existing records in the linked table.
Permissions: Navigate	• **True:** Allow users to navigate between records in the linked table. • **False:** Prevent users from navigating between records in the linked table.

Tip For more information on setting appropriate permissions, see Optimizing user permissions on page 560**.**

Working with menus

Applications use menu options to provide added functionality or alternative means of accessing features. Menu Items and Button controls provide the same types of features, including actions and table filters. MobileApp Designer provides a complete set of menu-building tools.

Working with Menubars

A Menubar is a top-level component on a menu that holds menu items. To add a Menubar to a project, select **Edit > Insert Menubar** from the menu or right-click the **Menus** root icon in the Workspace palette and select **Insert Menubar** from the context menu. To delete a form from a project, right-click the Menubar's icon in the Workspace palette and select **Delete Menubar** from the context menu.

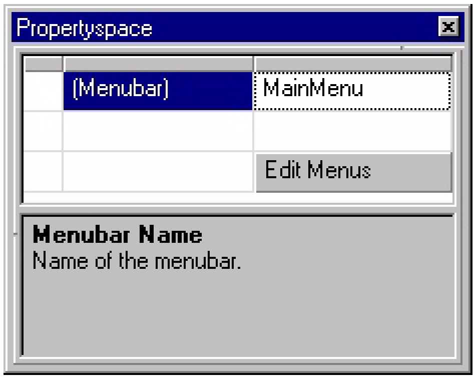
Menubar properties

This section describes the properties available for Menubars.

Note The **Menubar Name** property of MobileApp Designer Menubars cannot contain spaces. Use underscores instead of spaces for this property.

The Propertyspace palette for a Menubar is shown in the following figure:

Figure 6.13 Menubar Propertyspace palette



Propertyspace

(Menubar) MainMenu

Edit Menus

Menubar Name
Name of the menubar.

The properties available for a Menubar are listed in the following table:

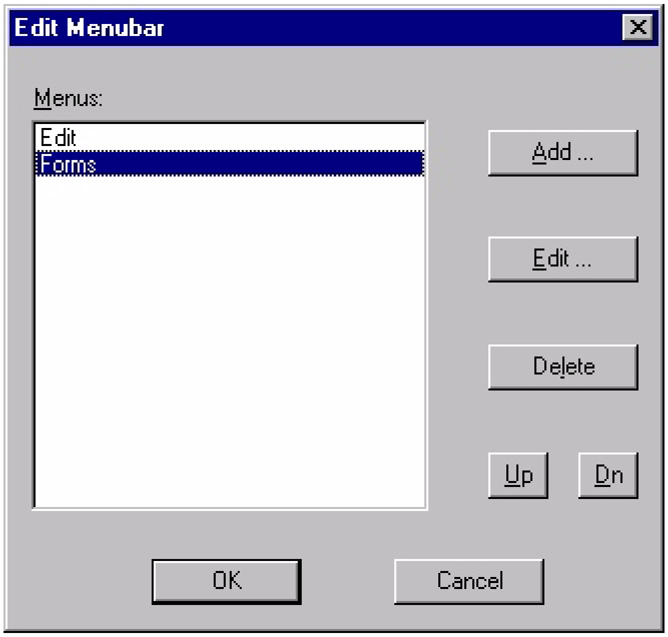
Table 6.4 Menubar properties

Property	Value
(Menubar)	The name of the Menubar. The current project refers to the Menubar by this name. Type the desired name.
Edit Menus	Click this button to add or edit the Menus associated with the selected Menubar, as described in the following section.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Adding or editing Menus

The Edit Menubar dialog box, as shown in the following figure, allows you to add or edit Menus attached to the selected Menubar.

Figure 6.14 Edit Menubar dialog box



- **Menus list:** Displays the Menus currently attached to the selected Menubar. Click the desired Menu to edit or delete it.
- **Add... button:** Adds a new Menu to the selected Menubar using the Menu Properties dialog box, as shown in Figure 6.16 on page 121.
- **Edit... button:** Edits the Menu selected in the Menus list using the Menu Properties dialog box, as shown in Figure 6.16 on page 121.
- **Delete button:** Deletes the Menu selected in the Menus list. MobileApp Designer does not prompt you to delete the selected Menu.
- **Up/Dn buttons:** To change the order of the Menus on the selected Menubar, click the Menu you would like to move and then click the **Up** or **Dn** button to position the Menu as desired. Repeat this process until the Menus are in the desired order.

Click the **OK** button to save all changes to the Menus in the selected Menubar. Click the **Cancel** button to close the Edit Menubar dialog box without saving changes to the selected Menubar.

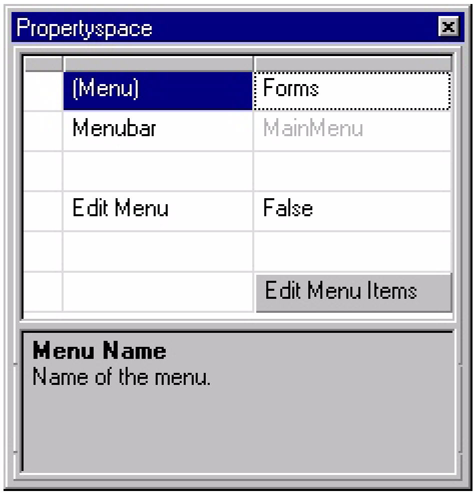
Menu Properties

This section describes the properties available for Menus.

Note The **Menu Name** property of MobileApp Designer Menus cannot contain spaces. Use underscores instead of spaces for this property.

The Propertyspace palette for a Menu is shown in the following figure:

Figure 6.15 Menu Propertyspace palette



(Menu)	Forms
Menubar	MainMenu
Edit Menu	False

Menu Name
Name of the menu.

The properties available for a Menu are listed in the following table:

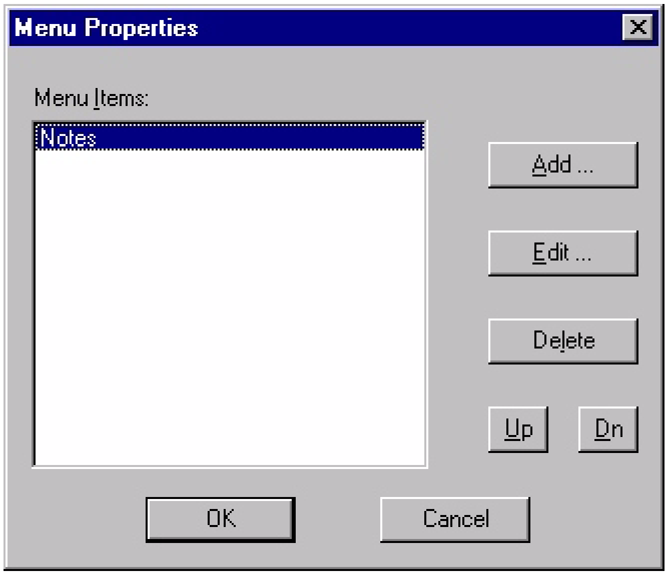
Table 6.5 Menu properties

Property	Value
(Menu)	The name of the Menu. The current project refers to the Menu by this name. Type the desired name.
Menubar	Read-only. Displays the name of the Menubar to which this Menu is attached.
Edit Menu	• **True:** The selected Menu is an OS-defined Edit menu. You cannot edit any other properties if you set this property to True. • **False:** The selected Menu is a custom menu.
Edit Menu Items	Click this button to add or edit the Menu Items associated with the selected Menu, as described in the following section.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Adding or editing Menu Items

The Edit Menu Items dialog box, as shown in the following figure, allows you to add or edit Menus attached to the selected Menubar.

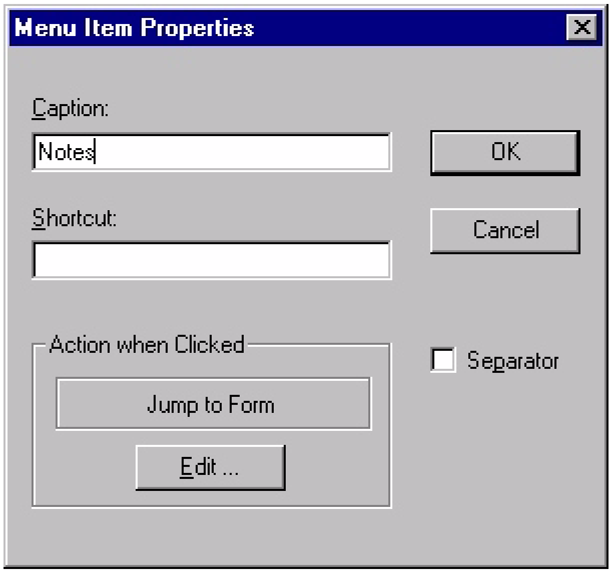
Figure 6.16 Menu Properties dialog box



- **Menu Items list:** Displays the Menu Items currently attached to the selected Menu. Click the desired Menu Item to edit or delete it.
- **Add... button:** Adds a new Menu Item to the selected Menu using the Menu Item Properties dialog box, as shown in Figure 6.17 on page 122.
- **Edit... button:** Edits the Menu Item selected in the Menu Items list using the Menu Item Properties dialog box, as shown in Figure 6.17 on page 122.
- **Delete button:** Deletes the Menu Item selected in the Menu Items list. MobileApp Designer does not prompt you to delete the selected Menu Item.
- **Up/Dn buttons:** To change the order of the Menu Items on the selected Menu, click the Menu Item you would like to move and then click the **Up** or **Dn** button to position the Menu Item as desired. Repeat this process until the Menu Items are in the desired order.

Click the **OK** button to save all changes to the Menu Items in the selected Menu. Click the **Cancel** button to close the Menu Properties dialog box without saving changes to the selected Menu.

Figure 6.17 Menu Item Properties dialog box



Caption: Notes

Shortcut:

Action when Clicked: Jump to Form

☐ Separator

- **Caption:** The text that appears on the Menu Item. Type the desired text.
- **Shortcut:** Shortcut key or keys for the selected Menu Item. Type the desired shortcut.
- **Separator:** When checked, designates this Menu Item as a separator line between groups of Menu Items, rather than an active Menu Item. You cannot set any other properties for the Menu Item if you check this box.
- **Action when Clicked:** Displays the action or filter for the Menu Item. See Setting up actions on page 177 for more information.
- **Edit:** Click this button to set the desired action or filter for the Menu Item. See Setting up actions on page 177 for more information.

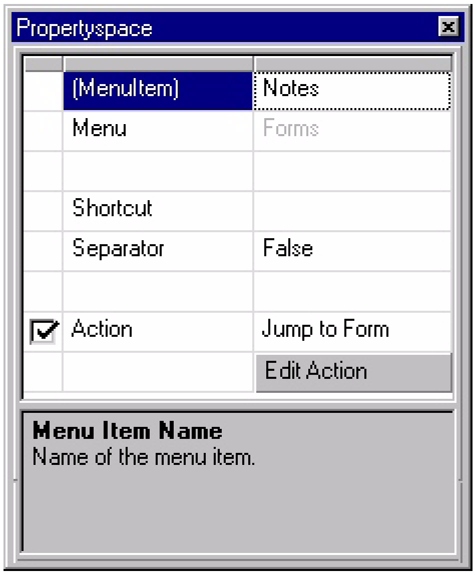
Menu Item Properties

This section describes the properties available for Menu Items.

Note The **MenuItem Name** property of MobileApp Designer Menu Items cannot contain spaces. Use underscores instead of spaces for this property.

The Propertyspace palette for a Menu Item is shown in the following figure:

Figure 6.18 Menu Item Propertyspace palette



	(MenuItem)	Notes
	Menu	Forms
	Shortcut	
	Separator	False
☑	Action	Jump to Form

Menu Item Name
Name of the menu item.

The properties available for a Menu Item are listed in the following table:

Table 6.6 Menu Item properties

Property	Value
(MenuItem)	The name of the Menu Item. The current project refers to the Menu Item by this name. Type the desired name.
Menu	Read-only. Displays the name of the Menu to which this Menu Item is attached.
Shortcut	Shortcut key or keys for the selected Menu Item. Type the desired shortcut.
Separator	• **True:** The Menu Item is a separator – a line dividing groups of Menu Items – not an active Menu Item. • **False:** The Menu Item is not a separator, but is instead an active Menu Item.
Action	Displays the action or filter for the Menu Item. See Setting up actions on page 177 for more information.
Edit Action	Click this button to set the desired action or filter for the Menu Item. See Setting up actions on page 177 for more information.

Table 6.6 Menu Item properties *(Continued)*

Property	Value
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Using controls

This section describes the available controls and their properties. Controls provide a wide range of functionality to your applications. All controls are associated with forms, although not all controls are visible on a form when the application is running on the handheld device.

The **Control Palette** toolbar, as shown in Figure 5.28 on page 97, provides one-click access to all of the available controls. The Control Palette toolbar becomes active whenever you select a form on the MobileApp Designer desktop.

To add a control to the selected form, click the desired button on the **Control Palette** toolbar. MobileApp Designer places the new control in the upper left-hand corner of the form, overlaying any control already in that position. Drag the new control to a different location to reveal the previously inserted control.

When you add a control to a form, an icon appears in the Workspace palette under the associated form's icon. Click a control to display its properties in the Propertyspace palette.

Working with controls

MobileApp Designer provides a variety of convenient ways of working with controls so that you can build applications that make efficient use of screen space and are efficient and easy to use.

To position a control on a form:

- Click and drag the control to the desired location on the form.
- Click the control you want to move and click the arrow keys on the keyboard to move it to the desired position.
- Click the control you want to position on the form and edit its Top and Left properties in the Propertyspace palette.

To set the size of a control:

- Click and drag the appropriate grapples until the control is the desired size. Grapples are the small black squares around the edge of the selected control.
- Click the control you want to resize and press **Ctrl** + the appropriate arrow key. The control's dimensions change by one pixel each time you press an arrow key.
- Click the control you want to resize and edit its Width and Height properties on the Propertyspace palette.

To set the z-order (stacking order) of two or more overlapping controls:

Click the desired control and then select **Edit > Bring Control to Front** or **Edit > Send Control to Back**. For more information, see MobileApp Designer menus on page 78.

To cut, copy, or delete one or more controls on a form:

- Click the desired control and then select **Edit > Cut Control**, **Edit > Copy Control**, or **Edit > Delete Control**. To cut, copy, or delete two or more controls on a form, press and hold the **Ctrl** key and click the desired controls. Then select the desired option from the Edit menu as usual. For more information, see MobileApp Designer menus on page 78.

To paste one or more controls from the clipboard onto a form:

- Click the form onto which you want to paste the control(s) on the clipboard and select **Edit > Paste Control**. For more information, see MobileApp Designer menus on page 78. Position and size the controls on the form as described above.

Control Properties

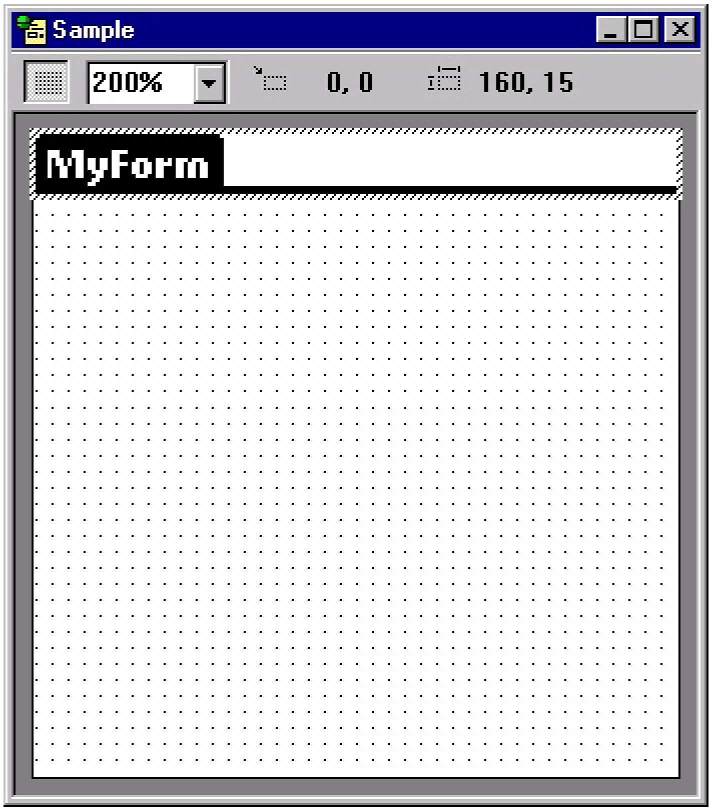
This section describes the properties available for each control MobileApp Designer provides.

Note The **Control Name** property of MobileApp Designer controls cannot contain spaces. Use underscores instead of spaces for this property.

Title control properties

The title control displays a title across the form or page, as shown in the following figure:

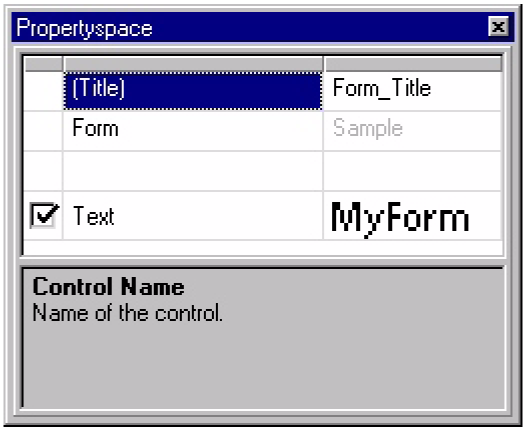
Figure 6.19 Form with Title control



Note The position of the Title control is fixed at the top of the form or page.

The Propertyspace palette for a Title control is shown in the following figure:

Figure 6.20 Title control Propertyspace palette



	(Title)	Form_Title
	Form	Sample
☑	Text	MyForm

Control Name
Name of the control.

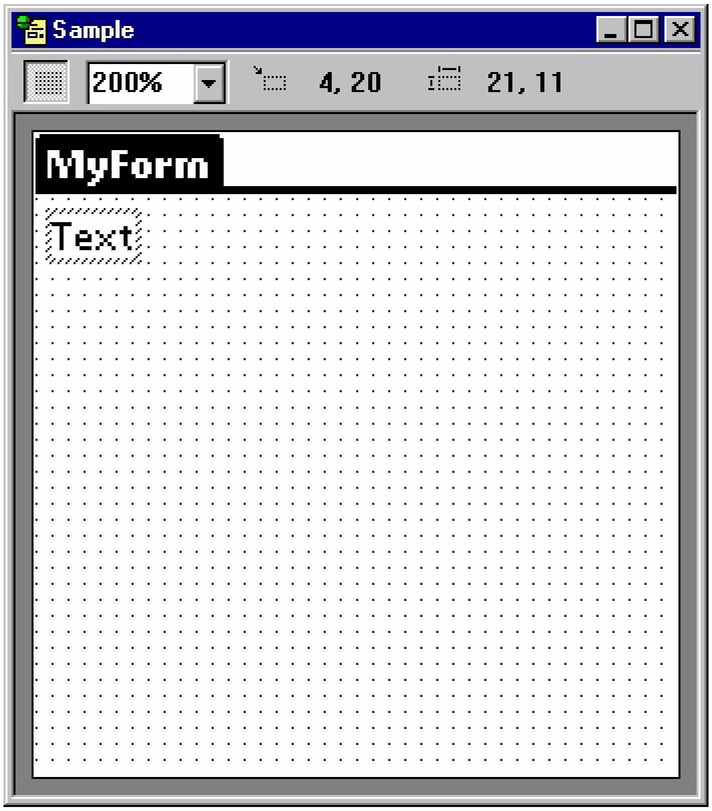
The properties available for a Title control are listed in the following table:

Table 6.7 Title control properties

Property	Value
(Title)	The name of the control. The current project refers to the control by this name. Type the desired name.
Text	The text that the Title control displays on the form or page. The display name of the form or page. Type the desired name.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Text control Text controls display static text on a form, as shown in the following figure:

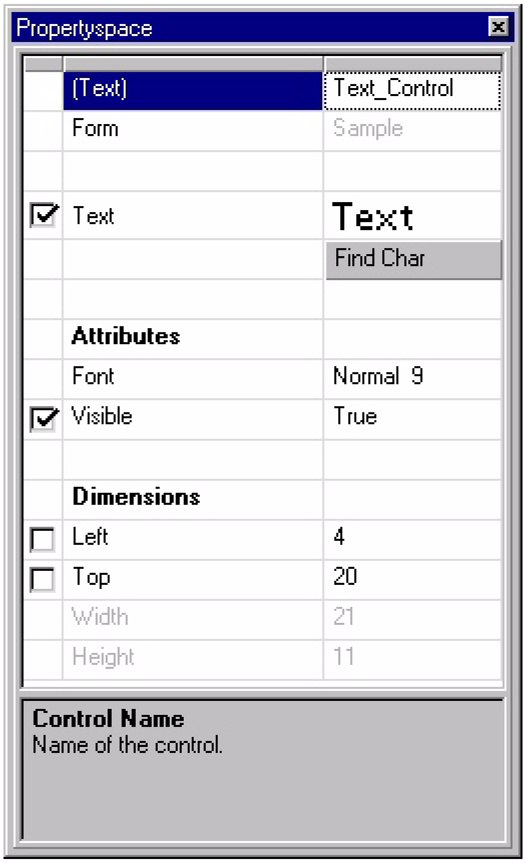
Figure 6.21 Form with Text control



Tip Use Text controls to label other controls, such as Edit and Paragraph controls, or to provide instructions on using a form or page.

The Propertyspace palette for a Text control is shown in the following figure:

Figure 6.22 Text control Propertyspace palette



Propertyspace

	(Text)	Text_Control
	Form	Sample
☑	Text	Text
		Find Char
	Attributes	
	Font	Normal 9
☑	Visible	True
	Dimensions	
□	Left	4
□	Top	20
	Width	21
	Height	11

Control Name
Name of the control.

The properties available for a Text control are listed in the following table:

Table 6.8 Text control properties

Property	**Value**
(Text)	The name of the control. The current project refers to the control by this name. Type the desired name.
Text	The text that the control displays on the form or page. Type the desired text. **Note:** Depending on the font you select for a Text control, the Text property may hold more characters than the handheld device can display. Use the appearance of the Text control in the Form design window as a guide to the maximum number of characters you can enter in a Text control.
Find Char	Click this button to display a table of the characters in the selected font. Click the character you want to insert at the carrot position in the Text property. Use this button to insert symbol characters into the Text property.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.

Table 6.8 Text control properties *(Continued)*

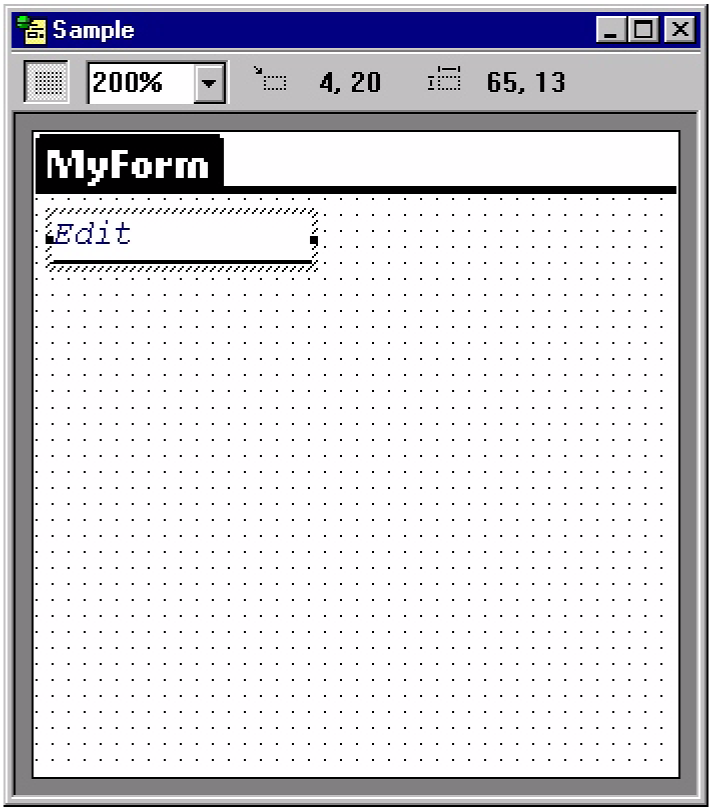
Property	Value
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Tip The Width and Height properties are set automatically based on the Font and Text properties.

Edit control

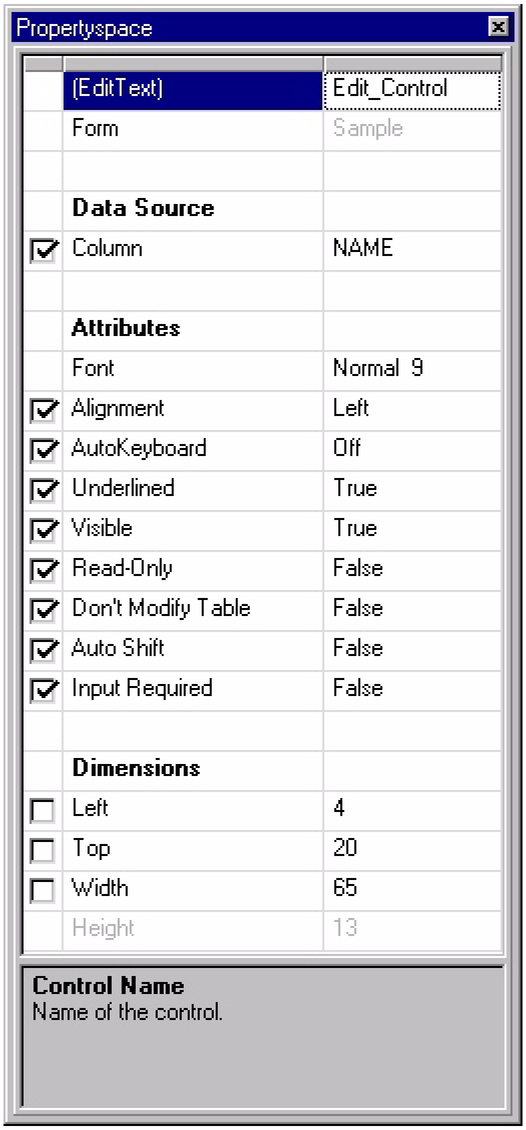
An Edit control, as shown in the following figure, displays the contents of the specified column for the current record of the form's linked table.

Figure 6.23 Form with Edit control



The Propertyspace palette for an Edit control is shown in the following figure:

Figure 6.24 Edit control Propertyspace palette



Propertyspace

	(EditText)	Edit_Control
	Form	Sample
	Data Source	
☑	Column	NAME
	Attributes	
	Font	Normal 9
☑	Alignment	Left
☑	AutoKeyboard	Off
☑	Underlined	True
☑	Visible	True
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Auto Shift	False
☑	Input Required	False
	Dimensions	
□	Left	4
□	Top	20
□	Width	65
	Height	13

Control Name
Name of the control.

The properties available for an Edit control are listed in the following table:

Table 6.9 Edit control properties

Property	**Value**
(EditText)	The name of the control. The current project refers to the control by this name. Type the desired name.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table. Select the desired column from the list.
Attributes: Font	Sets the font for the control. Select the desired font from the list.
Attributes: Alignment	Sets the text justification for the control. Select **Left** or **Right** from the list.

Table 6.9 Edit control properties *(Continued)*

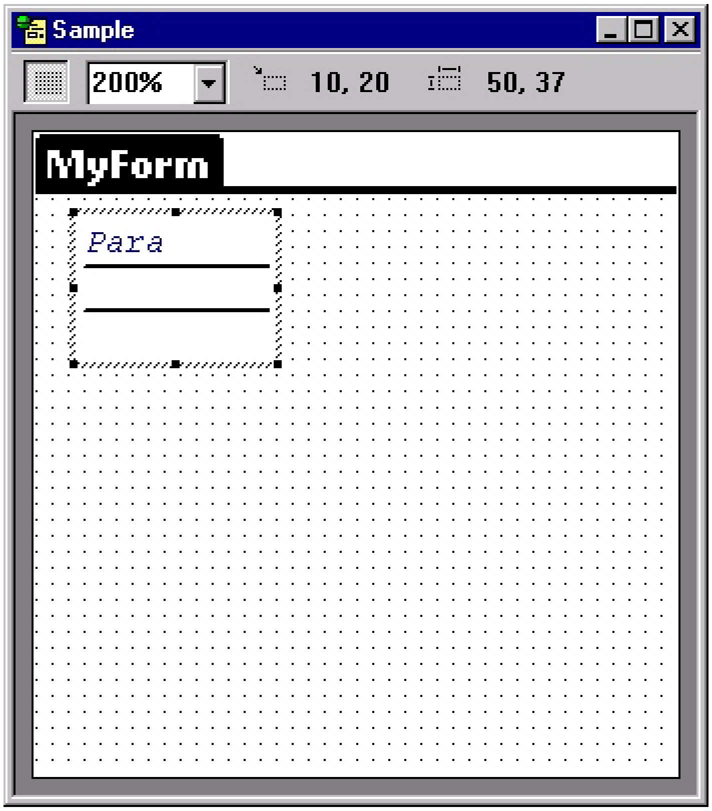
Property	**Value**
Attributes: AutoKeyboard	Sets the auto keyboard option for the control. Select the desired option from the list. This attribute has five possible values: • **Off:** When the user taps the edit control, no special handheld keyboard appears. • **Auto:** When the user taps the edit control, the appropriate handheld keyboard for the control's data source appears. • **Character:** When the user taps the edit control, the character handheld keyboard appears. • **Numeric:** When the user taps the edit control, the numeric handheld keyboard appears. • **Time:** When the user taps the edit control, the time entry handheld keyboard appears.
Attributes: Underlined	• **True:** Text typed into control is underlined. • **False:** Text typed into control is not underlined.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Read-Only	• **True:** The user cannot modify the text in the control. • **False:** The user can modify the text in the control.
Attributes: Don't Modify Table	• **True:** The user cannot modify the data in the linked column using this control. • **False:** The user can modify the data in the linked column using this control.
Attributes: Auto Shift	• **True:** The first character the user enters into the control is automatically capitalized. • **False:** The shift status of the first character the user enters into the control is not altered.
Attributes: Input Required	• **True:** The user cannot move to the next record or to another form before entering data into the control. • **False:** The user is not required to enter data into the control.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Tip When the Satellite Forms Engine displays a form, it places the cursor on the front-most Edit or Paragraph control on the form. To specify a particular Edit or Paragraph control as the front-most control on the form, select the control and then select **Edit > Bring Control to Front** from the MobileApp Designer menu.

Paragraph control

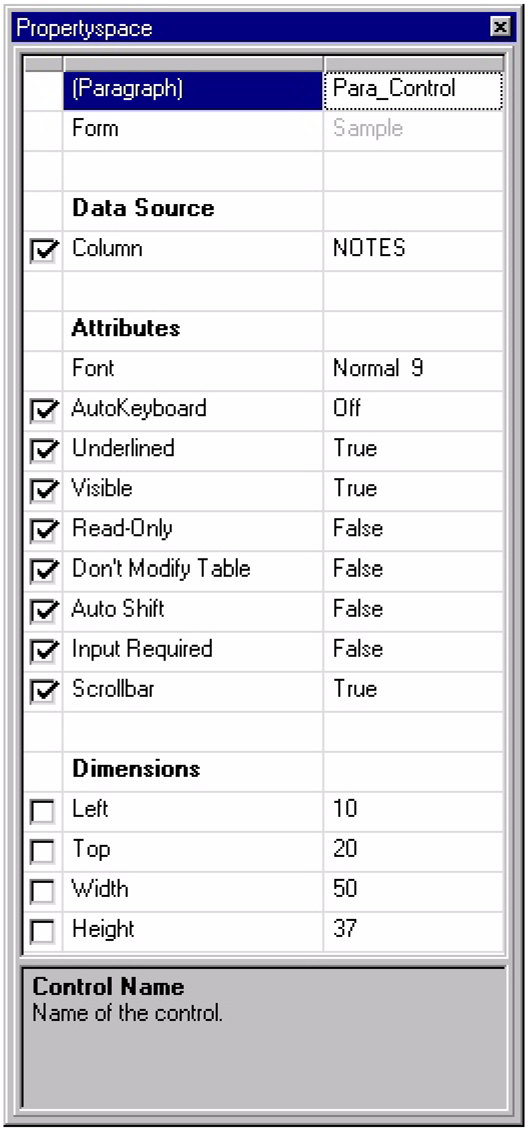
The Paragraph control, as shown in the following figure, is similar to the Edit control except that it displays information is displayed on multiple lines. If there is more information than the Paragraph control can display all at one time, a vertical scroll bar allows users to scroll to see the full contents of the control.

Figure 6.25 Form with Paragraph control



The Propertyspace palette for a Paragraph control is shown in the following figure:

Figure 6.26 Paragraph control Propertyspace palette



Propertyspace

	(Paragraph)	Para_Control
	Form	Sample
	Data Source	
☑	Column	NOTES
	Attributes	
	Font	Normal 9
☑	AutoKeyboard	Off
☑	Underlined	True
☑	Visible	True
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Auto Shift	False
☑	Input Required	False
☑	Scrollbar	True
	Dimensions	
□	Left	10
□	Top	20
□	Width	50
□	Height	37

Control Name
Name of the control.

The properties available for a Paragraph control are listed in the following table:

Table 6.10 Paragraph control properties

Property	Value
(Paragraph)	The name of the control. The current project refers to the control by this name. Type the desired name.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table. Select the desired column from the drop list.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.

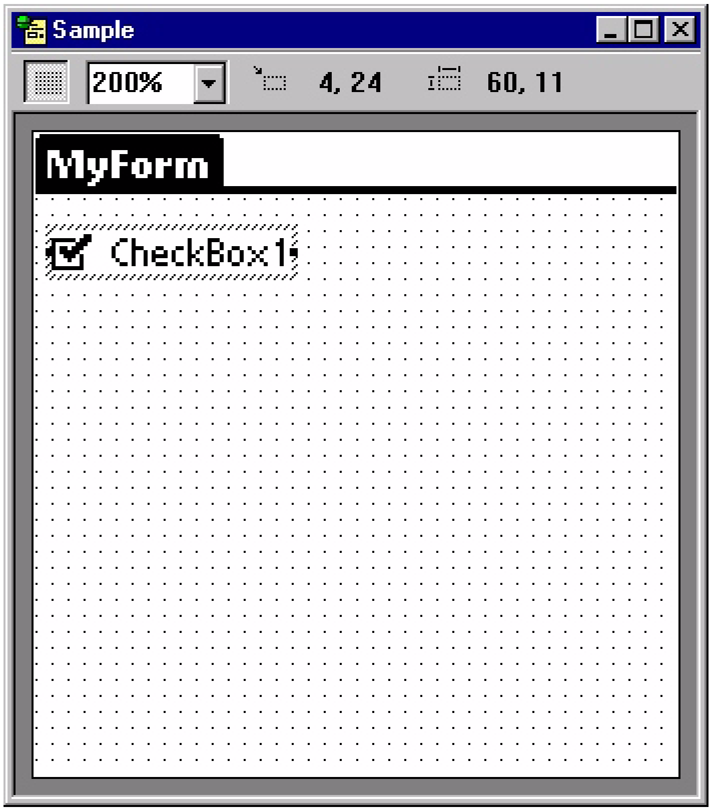
Table 6.10 Paragraph control properties *(Continued)*

Property	**Value**
Attributes: AutoKeyboard	Sets the auto keyboard option for the control. Select the desired option from the drop list. This attribute has five possible values: • **Off:** When the user taps the edit control, no special handheld keyboard appears. • **Auto:** When the user taps the edit control, the appropriate handheld keyboard for the control's data source appears. • **Character:** When the user taps the edit control, the character handheld keyboard appears. • **Numeric:** When the user taps the edit control, the numeric handheld keyboard appears. • **Time:** When the user taps the edit control, the time entry handheld keyboard appears.
Attributes: Underlined	• **True:** Text typed into control is underlined. • **False:** Text typed into control is not underlined.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Read-Only	• **True:** The user cannot modify the text in the control. • **False:** The user can modify the text in the control.
Attributes: Don't Modify Table	• **True:** The user cannot modify the data in the linked column using this control. • **False:** The user can modify the data in the linked column using this control.
Attributes: Auto Shift	• **True:** The first character the user enters into the control is automatically capitalized. • **False:** The shift status of the first character the user enters into the control is not altered.
Attributes: Input Required	• **True:** The user cannot move to the next record or to another form before entering data into the control. • **False:** The user is not required to enter data into the control.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Check Box control

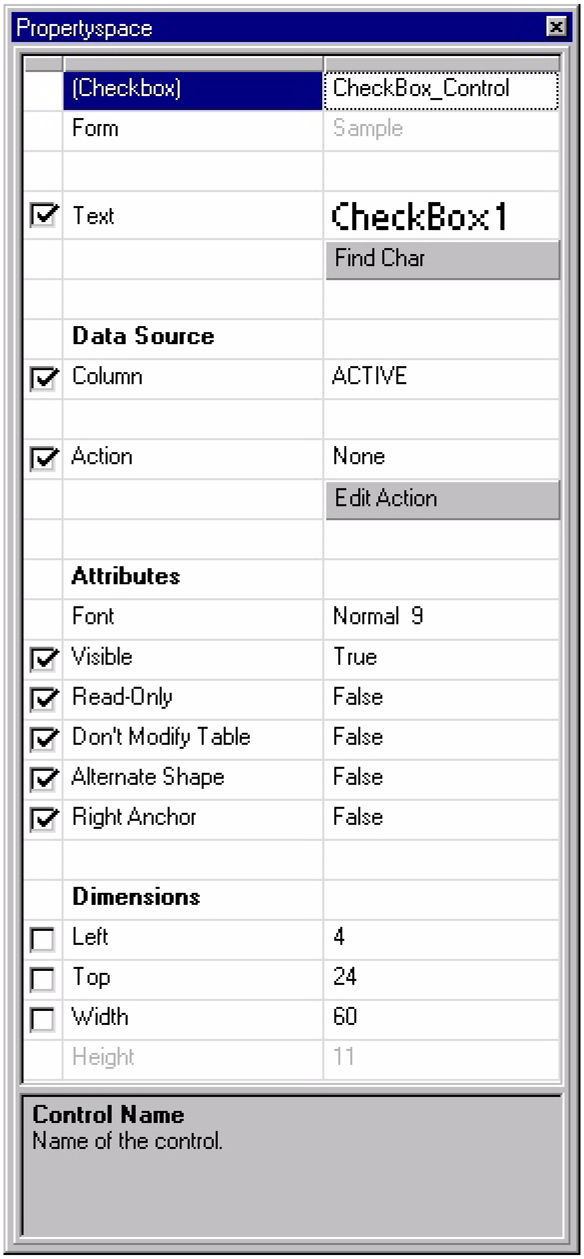
The Check Box control, as shown in the following figure, displays and optionally edits a True/False field from the current record of the form's linked table: checked equals a field value of True, cleared equals a field value of False.

Figure 6.27 Form with Check Box control



The Propertyspace palette for a Check Box control is shown in the following figure:

Figure 6.28 Check Box control Propertyspace palette



Propertyspace

	(Checkbox)	CheckBox_Control
	Form	Sample
☑	Text	CheckBox1
		Find Char
	Data Source	
☑	Column	ACTIVE
☑	Action	None
		Edit Action
	Attributes	
	Font	Normal 9
☑	Visible	True
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Alternate Shape	False
☑	Right Anchor	False
	Dimensions	
☐	Left	4
☐	Top	24
☐	Width	60
	Height	11

Control Name
Name of the control.

The properties available for a Check Box control are listed in the following table:

Table 6.11 Check Box control properties

Property	Value
(Checkbox)	The name of the control. The current project refers to the control by this name. Type the desired name.
Text	The text that the control displays on the form or page. Type the desired text.

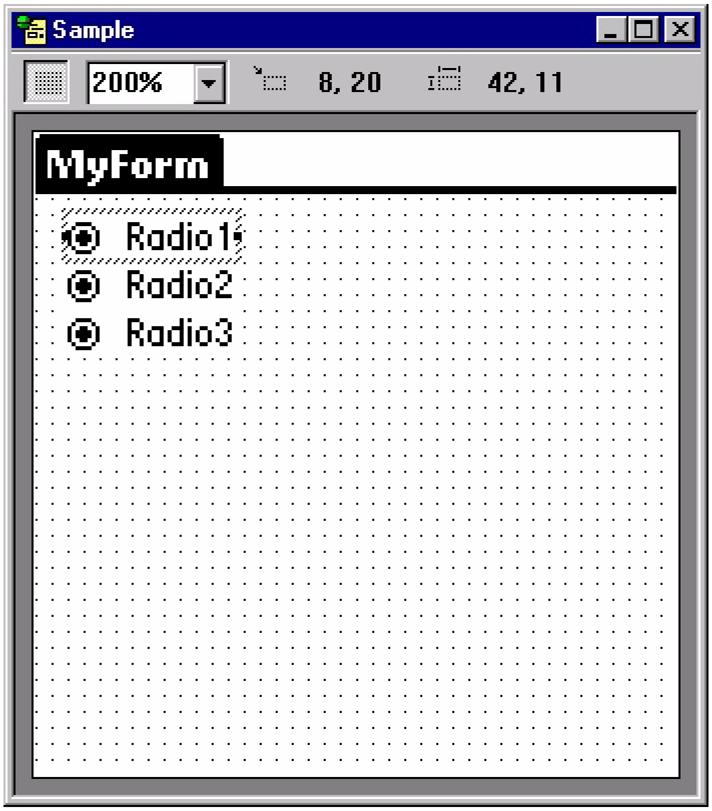
Table 6.11 Check Box control properties *(Continued)*

Property	Value
Find Char	Click this button to display a table of the characters in the selected font. Click the character you want to insert at the carrot position in the Text property. Use this button to insert symbol characters into the Text property.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table or if the linked table does not contain at least one True/False field. Select the desired column from the drop list.
Action	Displays the action or filter for the control. See Setting up actions on page 177 for more information.
Edit Action	Click this button to set the desired action or filter for the control. See Setting up actions on page 177 for more information.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Read-Only	• **True:** The user cannot modify the setting of the control. • **False:** The user can modify the setting of the control.
Attributes: Don't Modify Table	• **True:** The user cannot modify the data in the linked column using this control. • **False:** The user can modify the data in the linked column using this control.
Attributes: Alternate Shape	• **True:** Displays the alternate Check Box style, which resembles a button that sticks when tapped on the handheld. • **False:** Displays the standard Check Box style.
Attributes: Right Anchor	• **True:** The upper-right corner of the control is the anchor point for the control. • **False:** The upper-left corner of the control is the anchor point for the control.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value. • This property is only available if **Alternate Shape** is **True**.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Radio Button control

The Radio Button control, as shown in the following figure, works like a Check Box control except that the selection in a group of controls linked to the same field in the form's table is mutually exclusive. Radio Button controls can only be linked to Numeric fields.

Figure 6.29 Form with Radio Button control

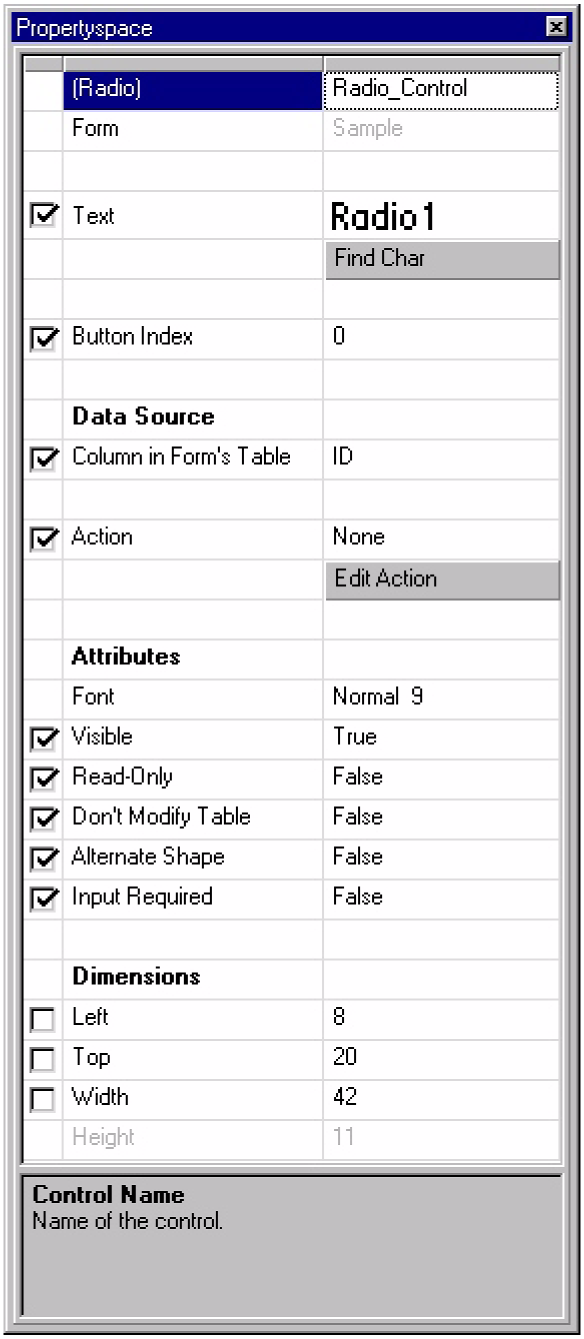


When more than one Radio Button control on a form uses the same column as its data source, these controls form a group. Only one of the controls in a group can be selected at a time. The value of the Button Index property is the integer placed in the linked Numeric field when the user selects the corresponding Radio Button. When the user navigates to a record with a field linked to one or more Radio Buttons, the Radio Button with the Button Index property corresponding to the value in the linked field becomes the selected button.

Note MobileApp Designer checks for previously used indices in the range of 0–31. If you use an index greater than 31, MobileApp Designer does not check to see if there are conflicts.

The Propertyspace palette for a Radio Button control is shown in the following figure:

Figure 6.30 Radio Button control Propertyspace palette



Propertyspace

	(Radio)	Radio_Control
	Form	Sample
☑	Text	Radio1
		Find Char
☑	Button Index	0
	Data Source	
☑	Column in Form's Table	ID
☑	Action	None
		Edit Action
	Attributes	
	Font	Normal 9
☑	Visible	True
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Alternate Shape	False
☑	Input Required	False
	Dimensions	
□	Left	8
□	Top	20
□	Width	42
	Height	11

Control Name
Name of the control.

The properties available for a Radio Button control are listed in the following table:

Table 6.12 Radio Button control properties

Property	Value
(Radio)	The name of the control. The current project refers to the control by this name. Type the desired name.

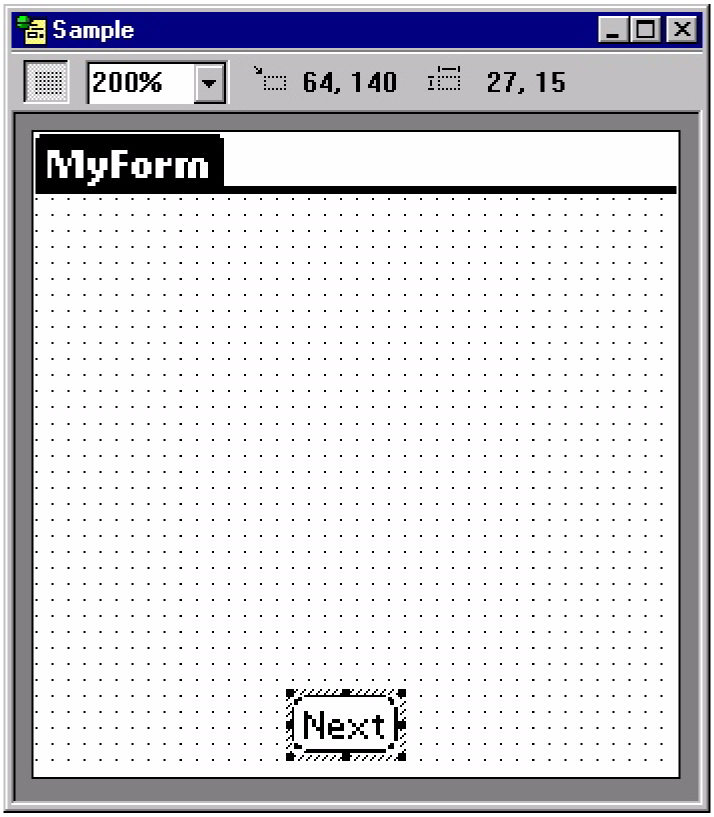
Table 6.12 Radio Button control properties *(Continued)*

Property	**Value**
Text	The text that the control displays on the form or page. Type the desired text.
Find Char	Click this button to display a table of the characters in the selected font. Click the character you want to insert at the carrot position in the Text property. Use this button to insert symbol characters into the Text property.
Button Index	The integer placed in the linked Numeric field when the user selects this Radio Button. When the user navigates to a record, the Radio Button with the Button Index property corresponding to the value in the linked field becomes the selected button. Select or type the desired index.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table or if the linked table does not contain at least one Numeric field. Select the desired column from the drop list.
Action	Displays the action or filter for the control. See Setting up actions on page 177 for more information.
Edit Action	Click this button to set the desired action or filter for the control. See Setting up actions on page 177 for more information.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Read-Only	• **True:** The user cannot modify the setting of the control. • **False:** The user can modify the setting of the control.
Attributes: Don't Modify Table	• **True:** The user cannot modify the data in the linked column using this control. • **False:** The user can modify the data in the linked column using this control.
Attributes: Alternate Shape	• **True:** Displays the alternate Radio Button style, which resembles a button that sticks when tapped on the handheld. • **False:** Displays the standard Radio Button style.
Attributes: Input Required	• **True:** The user cannot move to the next record or to another form before entering data into the control. • **False:** The user is not required to enter data into the control.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Button control

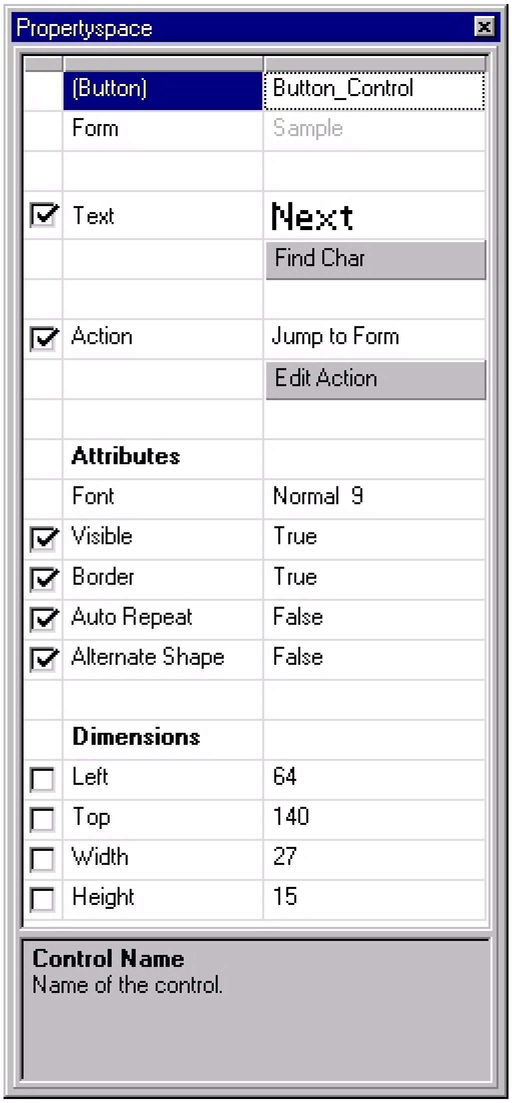
The Button control, as shown in the following figure, performs an action, such as jumping to another form, returning to the previous form, or creating a new record.

Figure 6.31 Form with Button control



The Propertyspace palette for a Button control is shown in the following figure:

Figure 6.32 Button control Propertyspace palette



Propertyspace

	(Button)	Button_Control
	Form	Sample
☑	Text	Next
		Find Char
☑	Action	Jump to Form
		Edit Action
	Attributes	
	Font	Normal 9
☑	Visible	True
☑	Border	True
☑	Auto Repeat	False
☑	Alternate Shape	False
	Dimensions	
□	Left	64
□	Top	140
□	Width	27
□	Height	15

Control Name
Name of the control.

The properties available for a Button control are listed in the following table:

Table 6.13 Button control properties

Property	Value
(Button)	The name of the control. The current project refers to the control by this name. Type the desired name.
Text	The text that the control displays on the form or page. Type the desired text.
Find Char	Click this button to display a table of the characters in the selected font. Click the character you want to insert at the carrot position in the Text property. Use this button to insert symbol characters into the Text property.
Action	Displays the action or filter for the control. See Setting up actions on page 177 for more information.

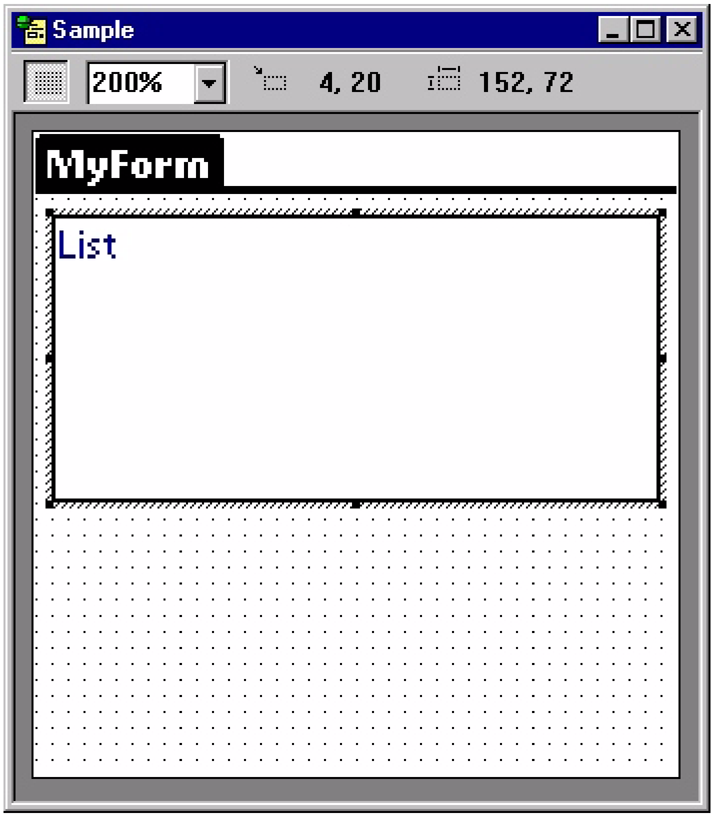
Table 6.13 Button control properties *(Continued)*

Property	**Value**
Edit Action	Click this button to set the desired action or filter for the control. See Setting up actions on page 177 for more information.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Border	• **True:** Draws a border around the Button control, giving it the standard appearance on the handheld device screen. • **False:** Removes the border from the Button control, displaying only the Text on the handheld device screen.
Attributes: Auto Repeat	• **True:** When the user taps the Button control, the Button fires its action immediately and continues to fire it at specified intervals if the user keeps the Button pressed. • **False:** When the user lifts the stylus from the Button, the Button fires its action.
Attributes: Alternate Shape	• **True:** Displays the alternate Button style, which is a square-edged rectangle. • **False:** Displays the standard Button style, which is a round-edged rectangle.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

List Box control

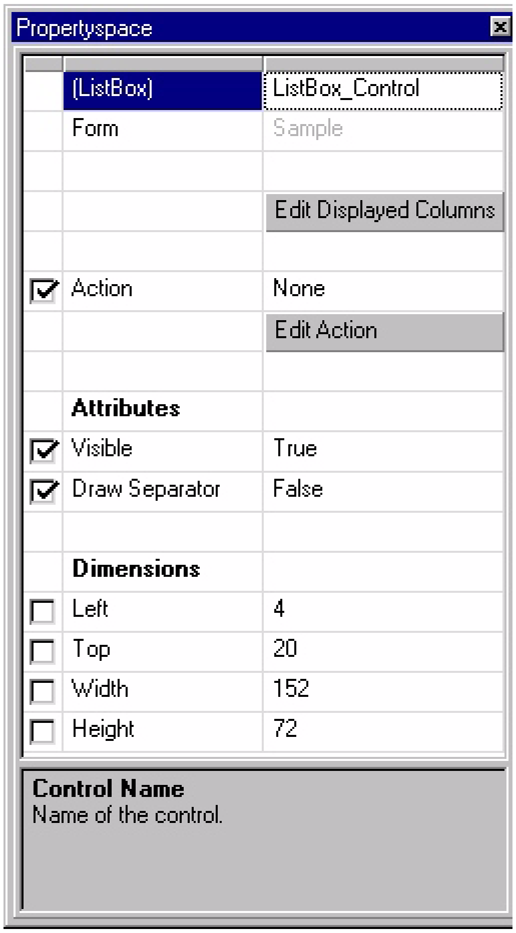
The List Box control, as shown in the following figure, displays multiple records from one or more columns of data from the form's linked table. If there are more fields than the List Box can display at one time, a vertical scroll bar allows the user to scroll to see all of the records.

Figure 6.33 Form with List Box control



The Propertyspace palette for a List Box control is shown in the following figure:

Figure 6.34 List Box control Propertyspace palette



The properties available for a List Box control are listed in the following table:

Table 6.14 List Box control properties

Property	Value
(ListBox)	The name of the control. The current project refers to the control by this name. Type the desired name.
Edit Displayed Columns	Click this button to set up the columns (fields) the List Box control displays. See Setting up a List Box control on page 147 for more information.
Action	Displays the action or filter for the control. See Setting up actions on page 177 for more information.
Edit Action	Click this button to set the desired action or filter for the control. See Setting up actions on page 177 for more information.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Draw Separator	• **True:** Draws a line between records displayed in the List Box control. • **False:** Displays records without separator lines.

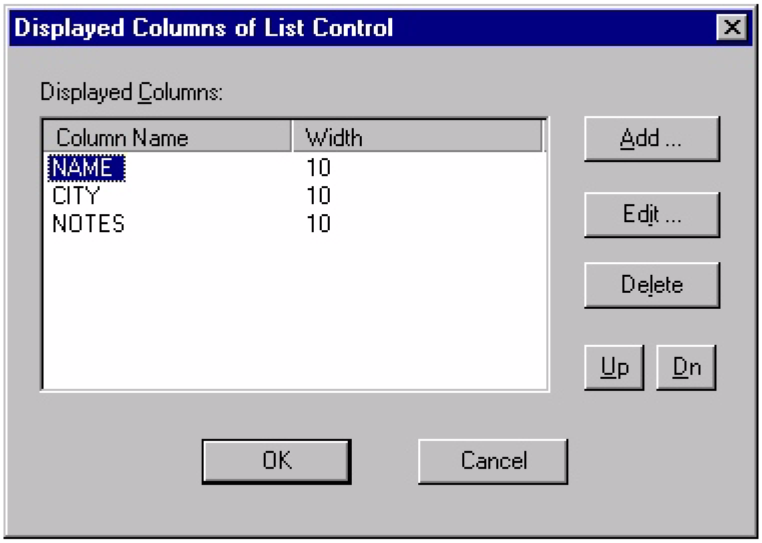
Table 6.14 List Box control properties *(Continued)*

Property	**Value**
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Setting up a List Box control

Click the **Edit Displayed Columns** button in the Propertyspace palette to set up a List Box control. The Displayed Columns of List Control dialog box shows the current column layout for the List Box control, as shown in the following figure:

Figure 6.35 Displayed Columns of List Control dialog box

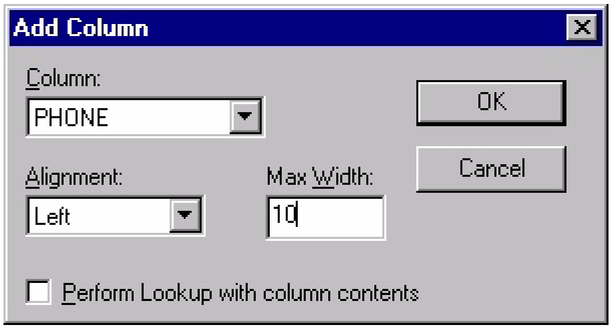


Column Name	Width
NAME	10
CITY	10
NOTES	10

To use this dialog box, follow these instructions:

- **Add...:** Click this button to display the **Add Column** dialog box, as shown in the following figure:

Figure 6.36 Add Column dialog box



Column: PHONE

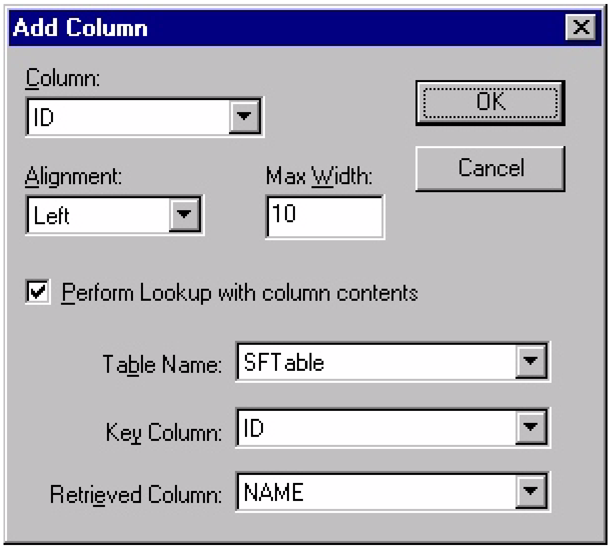
Alignment: Left

Max Width: 10

☐ Perform Lookup with column contents

- **Column:** Select the column to add using this combo box.
- **Alignment:** Select **Left** or **Right** from this combo box to set the justification of the data in the column.
- **Max Width:** Set the maximum width of the column in characters.
- **Perform Lookup with column contents:** Check this box to display the table lookup settings for the column, as shown in the following figure:

Figure 6.37 Add Column Dialog Box with Perform Lookup box checked



Column: ID

Alignment: Left

Max Width: 10

☑ Perform Lookup with column contents

Table Name: SFTable

Key Column: ID

Retrieved Column: NAME

The List Box control allows lookups to be performed on each column. A lookup matches a value in one table with the same value in another table thereby displaying more useful information in the List Box. For example, if the customers in a table are listed according to their ID numbers rather than their names, a lookup table contains the account numbers and the corresponding customer names can provide a useful lookup. If you specify a lookup on the ID column of the original table, the List Box control looks up the ID numbers listed in that column, matches them against the ID column in the lookup table, retrieves the appropriate customer name from the NAME column of the lookup table, and displays it on the List Box control.

To set up a lookup on the selected column, use these controls:

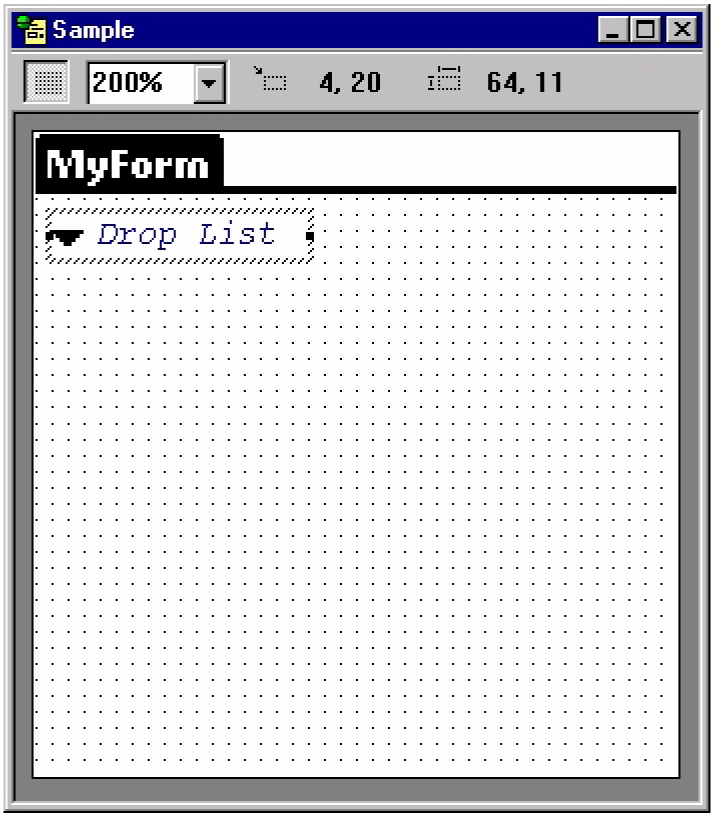
- **Table Name:** Select the table containing the desired lookup data using this combo box.
- **Key Column:** Select the key column using this combo box. This is the column in the lookup table that is matched with the column from the form's linked table.
- **Retrieved Column:** Select the column in the lookup table that is displayed on the form using this combo box.

- **Edit...:** To edit an existing column in the Displayed Columns of List Control dialog box, select the desired column and click the **Edit...** button. The Edit Column dialog box works exactly like the Add Column dialog box described above.
- **Delete:** To delete a column from the List Box control, select the column and then click the **Delete** button.
- **Up/Dn**: To change the order in which columns are displayed on the List Box control, use these two buttons to arrange the selected columns as desired. Select the column you want to move and then click the **Up** or **Dn** button until it is in the desired position.

Drop List control

The Drop List control, as shown in the following figure, displays a list of items from which a user can choose. A Drop List control is the space-saving equivalent of a Lookup control.

Figure 6.38 Form with Drop List control



The Drop List control's major advantage include space-saving and flexibility. When a Drop List is not expanded, it occupies only the space needed to display the current line. When a user taps the Drop List control, it expands to display a list from its linked table and column.

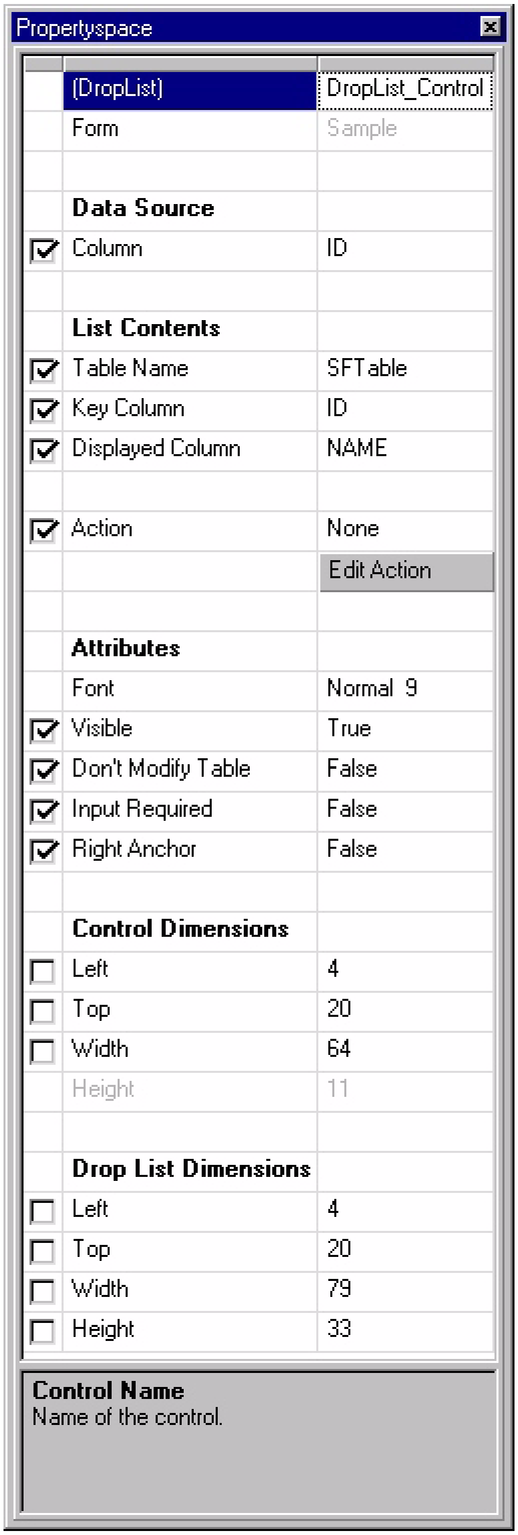
The items the Drop List control displays are derived from a column in a lookup table. The items displayed can be accessed indirectly through a link to a key column in the lookup table. For example, to display customer names in a Drop List control when the lookup table sorts names by customer ID number, set the Drop List's **Key Column** property to the ID number column and the **Displayed Column** property to the customer name column.

If you do not need the Drop List control to perform a lookup, set both the **Key Column** and the **Displayed Column** properties to the same column. Then, the Drop List control displays and stores the same values.

Tip Using filters, you can link two drop lists so that one shows a category and the other shows only items in the selected category. For details, see Linking Drop List controls on page 557.

The Propertyspace palette for a Drop List control is shown in the following figure:

Figure 6.39 Drop List control Propertyspace palette



	(DropList)	DropList_Control
	Form	Sample
	Data Source	
☑	Column	ID
	List Contents	
☑	Table Name	SFTable
☑	Key Column	ID
☑	Displayed Column	NAME
☑	Action	None
		Edit Action
	Attributes	
	Font	Normal 9
☑	Visible	True
☑	Don't Modify Table	False
☑	Input Required	False
☑	Right Anchor	False
	Control Dimensions	
□	Left	4
□	Top	20
□	Width	64
	Height	11
	Drop List Dimensions	
□	Left	4
□	Top	20
□	Width	79
□	Height	33

Control Name
Name of the control.

The properties available for a Drop List control are listed in the following table:

Table 6.15 Drop List control properties

Property	Value
(DropList)	The name of the control. The current project refers to the control by this name. Type the desired name.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table. Select the desired column from the list. This column is looked up in the Key Column of the lookup table.
List Contents: Table Name	The table containing the lookup information. The lookup table must be a part of the current project. Select the desired table from the list.
List Contents: Key Column	The column in both the form's linked table and the lookup table. This column must be of the same type and have the same name as the Data Source: Column property. Select the desired column from the list.
List Contents: Displayed Column	The column in the lookup table that the control displays. Select the desired column from the list.
Action	Displays the action or filter for the control. See Setting up actions on page 177 for more information.
Edit Action	Click this button to set the desired action or filter for the control. See Setting up actions on page 177 for more information.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Don't Modify Table	• **True:** The user cannot modify the data in the linked column using this control. • **False:** The user can modify the data in the linked column using this control.
Attributes: Input Required	• **True:** The user cannot move to the next record or to another form before selecting data using the control. • **False:** The user is not required to enter data into the control.
Attributes: Right Anchor	• **True:** The upper-right corner of the control is the anchor point for the control. • **False:** The upper-left corner of the control is the anchor point for the control.
Control Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Control Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Control Dimensions: Width	The width (in pixels) of the control. Type the desired value
Drop List Dimensions: Left	The left origin (in pixels) of the drop list part of the control. Type the desired value.
Drop List Dimensions: Top	The top origin (in pixels) of the drop list part of the control. Type the desired value.

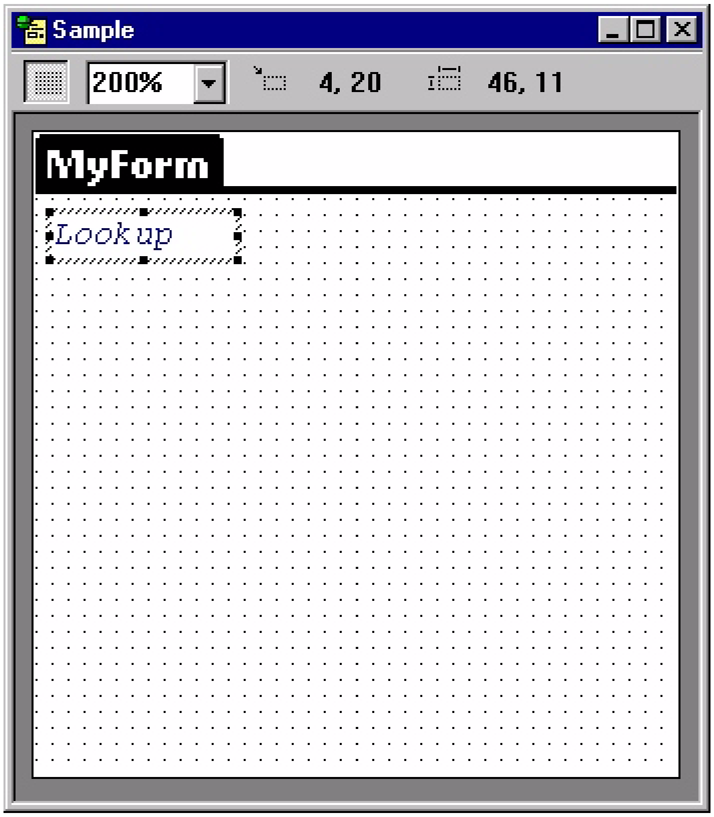
Table 6.15 Drop List control properties *(Continued)*

Property	Value
Drop List Dimensions: Width	The width (in pixels) of the drop list part of the control. Type the desired value.
Drop List Dimensions: Height	The height (in pixels) of the drop list part of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Lookup control

The lookup control, as shown in the following figure, displays a list of items from which a user can choose. Use Lookup Controls to display information in a user-friendly format by using a linked lookup table to retrieve information from another table that contains the data.

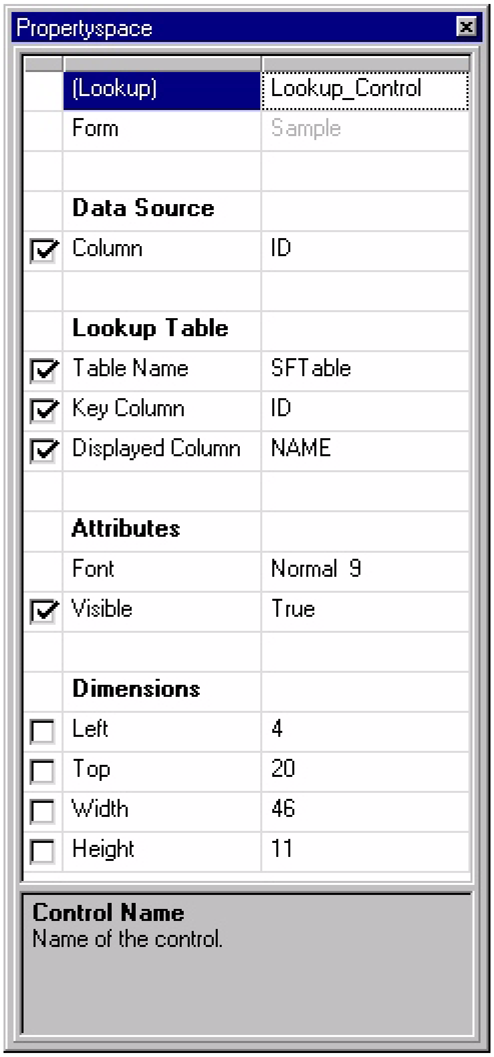
Figure 6.40 Form with Lookup control



The items the Lookup control displays are derived from a column in a lookup table. The items displayed can be accessed indirectly through a link to a key column in the lookup table. For example, to display customer names in a Lookup control when the lookup table sorts names by customer ID number, set the Lookup's **Key Column** property to the ID number column and the **Displayed Column** property to the customer name column.

The Propertyspace palette for a Lookup control is shown in the following figure:

Figure 6.41 Lookup control Propertyspace palette



	(Lookup)	Lookup_Control
	Form	Sample
	Data Source	
☑	Column	ID
	Lookup Table	
☑	Table Name	SFTable
☑	Key Column	ID
☑	Displayed Column	NAME
	Attributes	
	Font	Normal 9
☑	Visible	True
	Dimensions	
□	Left	4
□	Top	20
□	Width	46
□	Height	11

Control Name
Name of the control.

The properties available for a Lookup control are listed in the following table:

Table 6.16 Lookup control properties

Property	Value
(Lookup)	The name of the control. The current project refers to the control by this name. Type the desired name.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table. Select the desired column from the list. This column is looked up in the Key Column of the lookup table.
Lookup Table: Table Name	The table containing the lookup information. The lookup table must be a part of the current project. Select the desired table from the list.

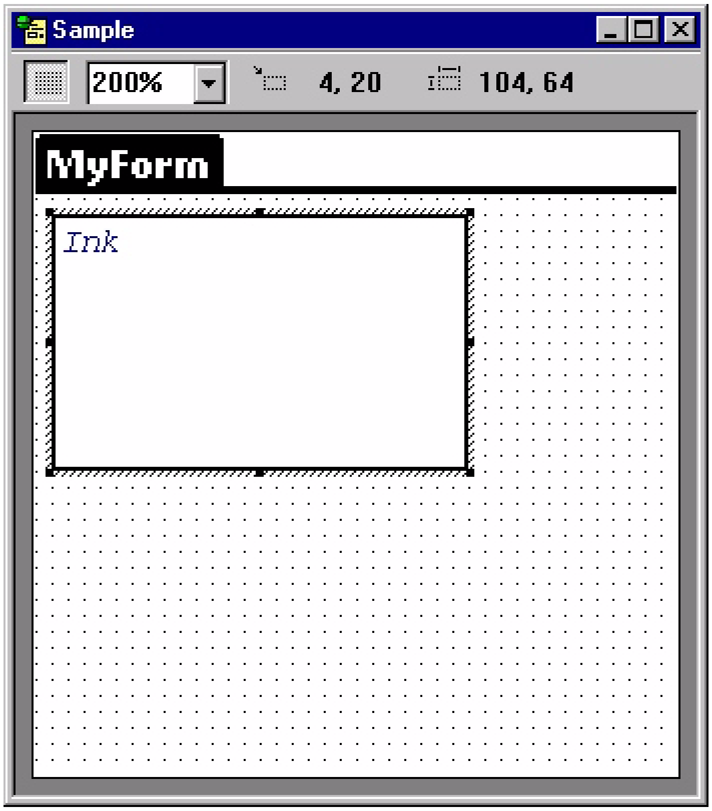
Table 6.16 Lookup control properties *(Continued)*

Property	Value
Lookup Table: Key Column	The column in both the form’s linked table and the lookup table. This column must be of the same type and have the same name as the Data Source: Column property. Select the desired column from the list.
Lookup Table: Displayed Column	The column in the lookup table that the control displays. Select the desired column from the list.
Attributes: Font	Sets the font for the control. Select the desired font from the drop list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Ink control

The ink control, as shown in the following figure, allows you to collect signatures or sketches from the handheld device. The drawing or signature created in the Ink control is compressed and saved in a Binary field designated as the control’s data source.

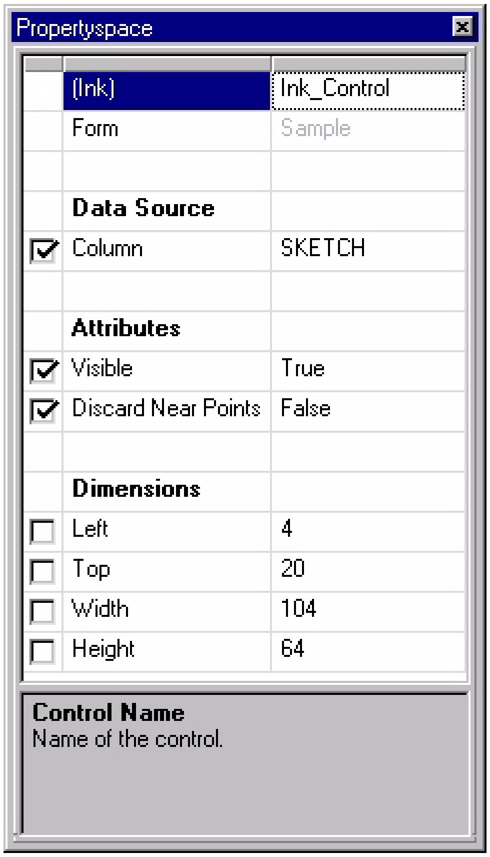
Figure 6.42 Form with Ink control



Note The contents of the Ink control's data source can only be transferred from the handheld device to MobileApp Designer or the application's associated DBMS, but not back to the handheld device.

The Propertyspace palette for an Ink control is shown in the following figure:

Figure 6.43 Ink control Propertyspace palette



Propertyspace

	(Ink)	Ink_Control
	Form	Sample
	Data Source	
☑	Column	SKETCH
	Attributes	
☑	Visible	True
☑	Discard Near Points	False
	Dimensions	
□	Left	4
□	Top	20
□	Width	104
□	Height	64

Control Name
Name of the control.

The properties available for an Ink control are listed in the following table:

Table 6.17 Ink control properties

Property	Value
(Ink)	The name of the control. The current project refers to the control by this name. Type the desired name.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table or if the linked table does not have at least one Binary field. Select the desired column from the list.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Attributes: Discard Near Points	• **True:** The Ink control smooths the drawing by discarding points that are within a preset distance of each other. Best for capturing sketches. • **False:** The Ink control saves all points in the drawing. Best for capturing signatures.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.

Table 6.17 Ink control properties *(Continued)*

Property	Value
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Procedure Inserting an Ink control

1 Make sure the form you want to add the Ink control to is the active form in the form design window.

2 Click the **Ink Control** button in the Control Palette.

3 The Ink View Control appears in the active form as a box with the words *Ink*.

4 If needed, reposition the ink control by clicking inside the control and dragging it into position.

5 If needed, resize the ink control to the desired width and height.

6 The ink control has been added to your form.

Tip Display the ink image captured on the handheld in your desktop database appllication using the SatForms Ink View OCX control. See the Deliveries sample application and KnowledgeBase for complete details.

Tip The InkHelper extension provides options to convert the binary ink data into other formats on the handheld, including a standard Windows BMP file. See the InkHelper article in the KnowledgeBase for more information.

Bitmap control

The bitmap control, as shown in the following figure, places a bitmap image on a form. Use this control to add pictures, logos, and other interesting visual touches to your forms.

Both color and black & white bitmaps are supported. Satellite Forms supports 5 bit depths: 1-bit, 2-bit, 4-bit, 8-bit and 16-bit. However, you must always supply a 1-bit (black and white) version of the image. This will ensure that your application will always look right on black and white devices.

Once you have associated a black and white image with a bitmap control, MobileApp Designer will use a predefined file naming convention to locate the other versions of the image file. MobileApp Designer will use the suffix "-2", "-4", "-8" and "-16" to locate the color versions of the black and white image. The number corresponds to the bit-depth of the file.

For example: if the bitmap control is associated with "Photo.bmp", MobileApp Designer will look for "Photo-2.bmp", "Photo-4.bmp", etc.

You don't have to specify images for all bit-depths. Often, supplying a black & white image plus an 8-bit (256 color) image is all you need to have your image look good on both monochrome and color devices. Higher bit images occupy a larger memory footprint. When building the project files, MobileApp Designer will use a dithering algorithm to transform 2, 4, 8 and 16-bit images so that they will use the correct Palm's color palettes, so your images may look different on the device. To avoid this transformation effect, you may want to use a graphic editing tool to make sure that the images are already using the correct Palm color palettes. For your reference, Satellite Forms installs "PalmPalettes.bmp" in the Templates directory.

You can use Microsoft Paint to create your images, if you do not have a more advanced image editor that you prefer to use. You can usually find this program in the Accessories submenu in the Windows Start menu.

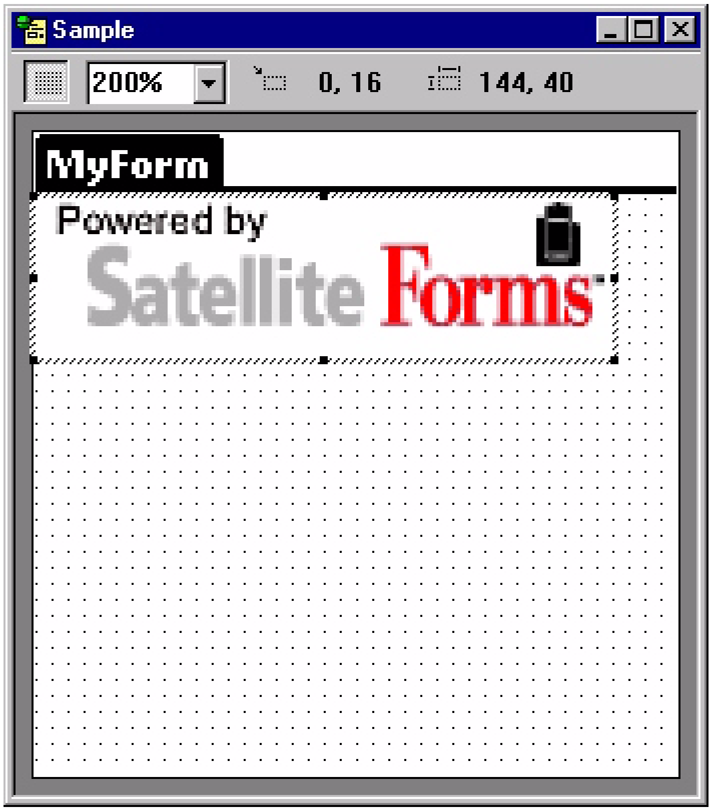
Note: using Microsoft Paint, 1-bit BMP file must be saved as Monochrome Bitmap and other BMP files must be saved as 24-bit Bitmap. See the about box in any of Satellite Forms sample projects for examples.

Starting with Satellite Forms 7.0, high density (HD) bitmaps can be used in your PalmOS applications. High density bitmaps occupy the same amount of space on the form, but display 4 times as much detail as standard density bitmaps, on PalmOS deveices that support high density displays. High density bitmaps are supported at the 8 and 16 bit color depths. To add an 8 bit high density image to your application, save the high density version of your bitmap with the -8-HD.bmp file suffix. To add a 16 bit high density image to your application, save the high density version of your bitmap with the -16-HD.bmp file suffix.

The HD versions of your bitmaps need to be exactly twice as wide and twice as tall as the standard density versions. For example, if a standard density bitmap is 44x43 pixels, the high density version of the image must be 88x86 pixels in size. MobileApp Designer will always display the standard density version of the image in the form designer view: it will not display the high density version of the image even if it is available.

Note: There is a PalmOS limit of 64K for each bitmap family (all bit depths combined).

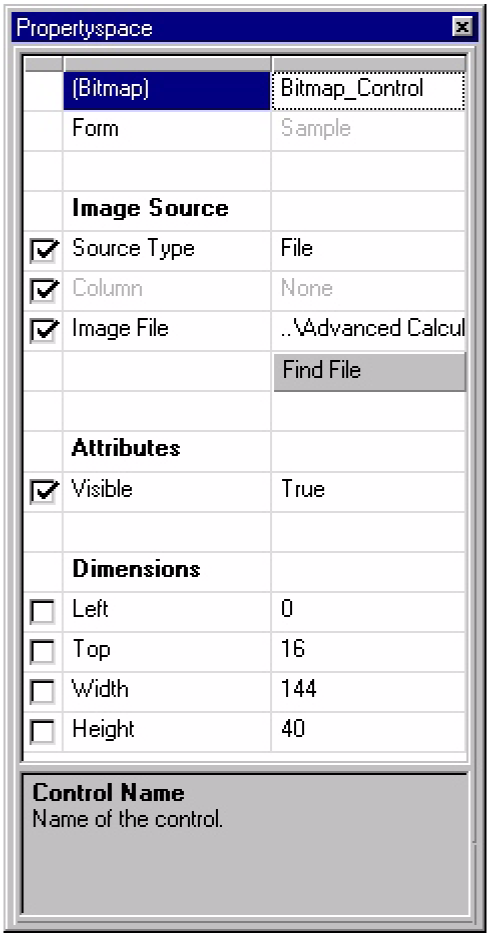
Figure 6.44 Form with Bitmap control



Tip Make sure the bitmap image you place with a Bitmap control is the desired size. The Bitmap control does not allow you scale the image, although you can crop an image by changing the size of the Bitmap control.

The Propertyspace palette for a Bitmap control is shown in the following figure:

Figure 6.45 Bitmap control Propertyspace palette



	(Bitmap)	Bitmap_Control
	Form	Sample
	Image Source	
☑	Source Type	File
☑	Column	None
☑	Image File	..\Advanced Calcul
		Find File
	Attributes	
☑	Visible	True
	Dimensions	
□	Left	0
□	Top	16
□	Width	144
□	Height	40

Control Name
Name of the control.

The properties available for a Bitmapcontrol are listed in the following table:

Table 6.18 Bitmap control properties

(Bitmap)	The name of the control. The current project refers to the control by this name. Type the desired name.
Image Source: Source Type	• **File:** The bitmap is loaded from a file on the development computer. • **Table:** The bitmap is loaded from a Binary field in the form's linked table.
Image Source: Column	The name of the column (field) in the form's linked table from which the image is loaded. This attribute is not available if the form on which it is placed does not have a linked table or if the linked table does not have at least one Binary field. Select the desired column from the list. Available only if **Source Type** is **Table**.
Image Source: Image File	The path and filename of the bitmap file the control displays. Available only if **Source Type** is **File**. This must be the black & white version of the bitmap: MobileApp Designer will locate the color version of the bitmap based on the naming conventions described above.
Image Source: Find File	Click this button to browse the development computer's hard drive for the desired bitmap image.

Table 6.18 Bitmap control properties *(Continued)*

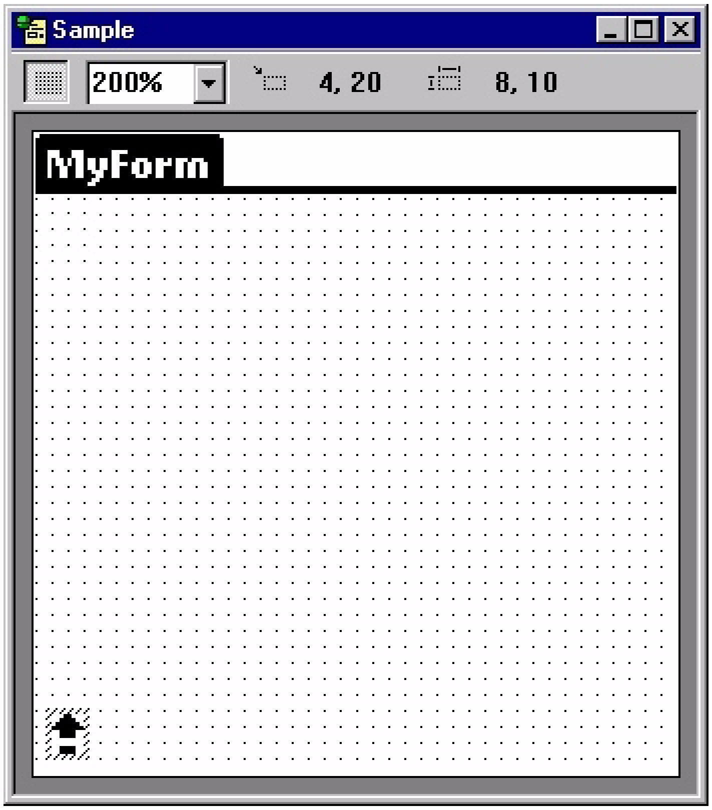
(Bitmap)	The name of the control. The current project refers to the control by this name. Type the desired name.
Attributes: Visible	• **True:** The control is visible on the form. • **False:** The control is not visible on the form.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Graffiti Shift Indicator control

The Graffiti Shift Indicator control, as shown in the following figure, places a graphic on the handheld device’s screen that shows the shift status of the Graffiti handwriting recognizer. This control displays different symbols for lowercase, shifted, or caps lock Graffiti mode.

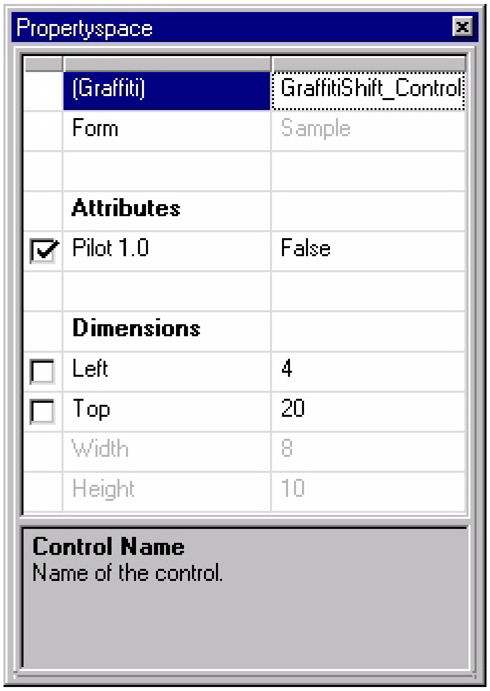
Note The Graffiti shift control is not available for Pocket PC applications.

Figure 6.46 Form with Graffiti Shift Indicator control



The Propertyspace palette for a Graffiti Shift Indicator control is shown in the following figure:

Figure 6.47 Graffiti Shift Indicator control Propertyspace palette



Propertyspace

	(Graffiti)	GraffitiShift_Control
	Form	Sample
	Attributes	
☑	Pilot 1.0	False
	Dimensions	
□	Left	4
□	Top	20
	Width	8
	Height	10

Control Name
Name of the control.

The properties available for a Graffiti Shift Indicator control are listed in the following table:

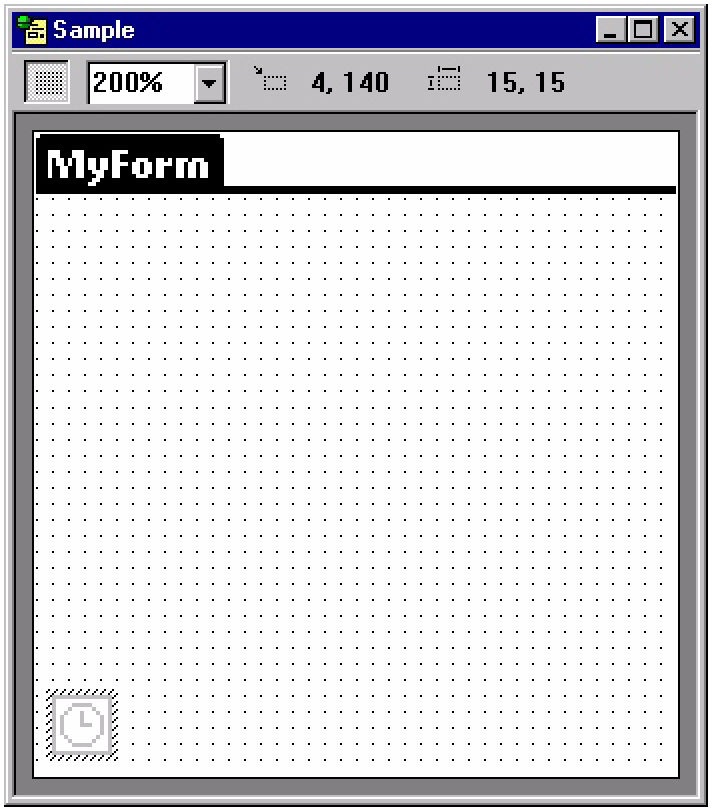
Table 6.19 Graffiti Shift Indicator control properties

Property	Value
(Graffiti)	The name of the control. The current project refers to the control by this name. Type the desired name.
Attributes: Pilot 1.0	• **True:** The control displays in Pilot 1.0 style. • **False:** The control displays in its standard style.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Auto Stamp control

The Auto Stamp control, as shown in the following figure, automatically enters a date or time stamp in a field.

Figure 6.48 Form with Auto Stamp control

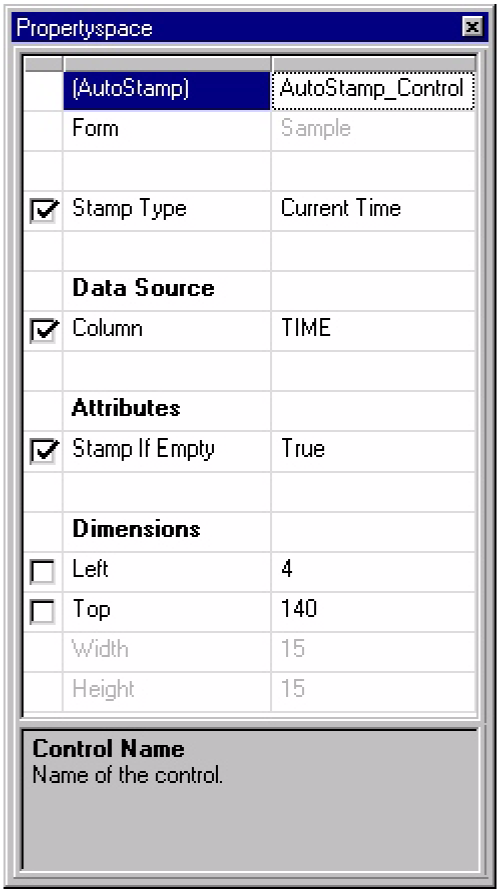


Tip The Auto Stamp control is invisible so place it anywhere on the form that is out of the way.

Tip Display the time/date stamp on the form by adding an Edit control linked to the same data source as the Auto Stamp control.

The Propertyspace palette for an Auto Stamp control is shown in the following figure:

Figure 6.49 Auto Stamp control Propertyspace palette



	(AutoStamp)	AutoStamp_Control
	Form	Sample
☑	Stamp Type	Current Time
	Data Source	
☑	Column	TIME
	Attributes	
☑	Stamp If Empty	True
	Dimensions	
□	Left	4
□	Top	140
	Width	15
	Height	15

Control Name
Name of the control.

The properties available for an Auto Stamp control are listed in the following table:

Table 6.20 Auto Stamp control properties

Property	Value
(AutoStamp)	The name of the control. The current project refers to the control by this name. Type the desired name.
Stamp Type	• **Current Date:** Places the current date in the Data Source column. The Data Source column must be a Date type. • **Current Time:** Places the current time, without seconds, in the Data Source column. The Data Source column must be a Time type. • **Current Time (including seconds):** Places the current time, with seconds, in the Data Source column. The Data Source column must be a Time type.
Data Source: Column	The name of the column (field) in the form's linked table to which this control is linked. This attribute is not available if the form on which it is placed does not have a linked table. Only columns of the type compatible with the **Stamp Type** are available. Select the desired column from the list.

Table 6.20 Auto Stamp control properties *(Continued)*

Property	Value
Attributes: Stamp If Empty	• **True:** The control stamps the date or time only if the linked field in the current record is empty. The control does not overwrite existing data. • **False:** The control stamps the date or time whether the linked field in the current record is empty or not. The control does overwrite existing data if present.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

SFX Custom control

An SFX Custom control is a program that adds functionality not provided by the standard controls to your Satellite Forms applications. You can add SFX Custom controls to the forms in your applications and use them as you would any of the standard MobileApp Designer controls.

Note This section uses the Slider control as an example of how to use SFX Custom controls in general. For information on a specific SFX Custom control, check the box to the left of the desired control and click the **Properties...** button in the Available Extensions dialog box, as shown in Figure 6.50 on page 167.

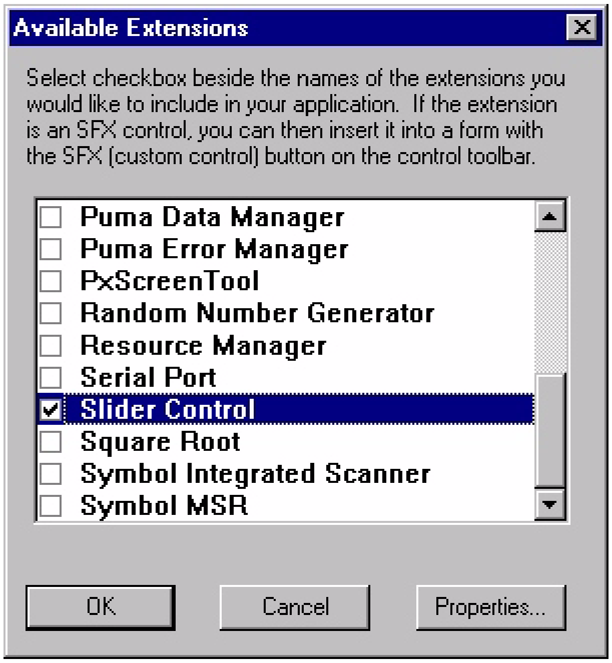
Satellite Forms installs several SFX Custom controls both for you to use in you applications and also to serve as examples of how to create your own custom controls. For more information on creating SFX Custom controls, see Chapter , Satellite Forms API Reference, on page 467

Note SFX Custom controls, such as the Slider control described in this section, are user interface objects. SFX plug-ins, as described under Adding extensions to Satellite Forms on page 188, are non-visual programs that provide added funtionality to scripts, such as mathematical functions.

Before you can add an SFX Custom control to a form, you must add the desired control to the project. To add an SFX Custom control to the current project, select **View > Extensions...** from the MobileApp Designer menu or click the **Manage Extensions** button on the Misc toolbar.

Use the **Available Extensions** dialog box, shown in the following figure, to select the Extension(s) to add to or remove from the current project by checking or clearing the check box to the left of each Extension name.

Figure 6.50 Available Extensions dialog box

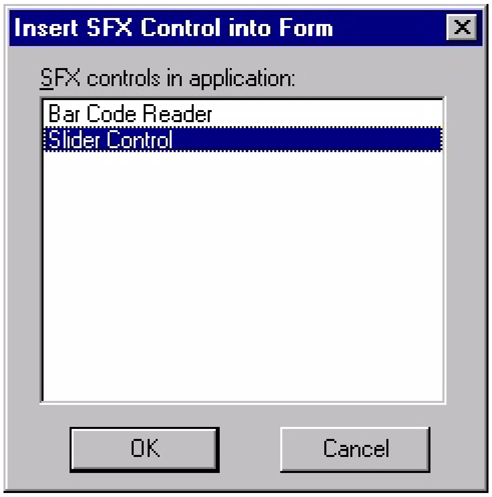


Select checkbox beside the names of the extensions you would like to include in your application. If the extension is an SFX control, you can then insert it into a form with the SFX (custom control) button on the control toolbar.

- □ Puma Data Manager
- □ Puma Error Manager
- □ PxScreenTool
- □ Random Number Generator
- □ Resource Manager
- □ Serial Port
- ☑ Slider Control
- □ Square Root
- □ Symbol Integrated Scanner
- □ Symbol MSR

To add an SFX Custom control to a form, click the **Custom Control** button SFX on the Control Palette toolbar, select the desired control from the **Insert SFX Control into Form** dialog box, shown in the following figure, and click the **OK** button.

Figure 6.51 Insert SFX Control into Form dialog box



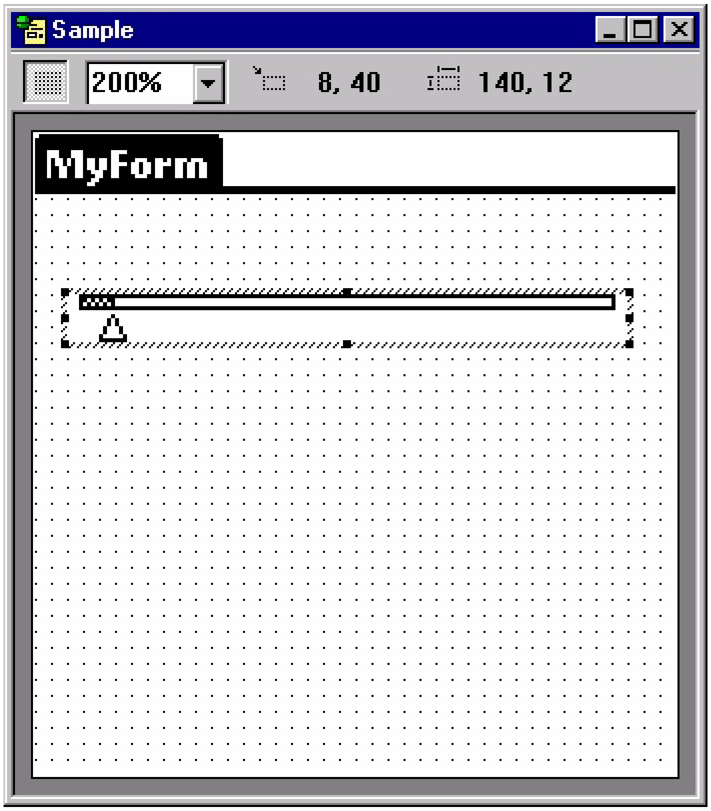
SFX controls in application:

- Bar Code Reader
- Slider Control

Note The Insert SFX Control into Form dialog box lists only the SFX Custom controls you have added to the current project.

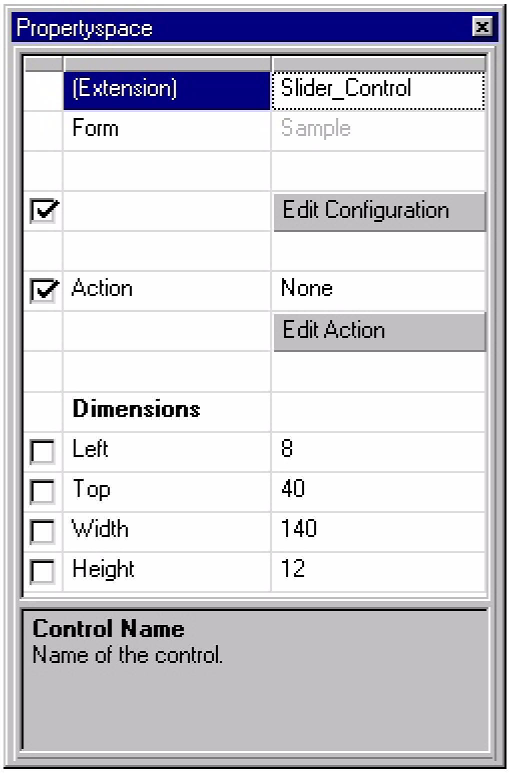
The following figure shows a form with a Slider control:

Figure 6.52 Form with Slider control



The Propertyspace palette for a Slider control is shown in the following figure:

Figure 6.53 Slider SFX Custom control Propertyspace palette



Propertyspace

	(Extension)	Slider_Control
	Form	Sample
☑		Edit Configuration
☑	Action	None
		Edit Action
	Dimensions	
□	Left	8
□	Top	40
□	Width	140
□	Height	12

Control Name
Name of the control.

The properties available for an Slider SFX Custom control are listed in the following table:

Table 6.21 Slider SFX Custom control properties

Property	**Value**
(Extension)	The name of the control. The current project refers to the control by this name. Type the desired name.
Edit Configuration	The configuration specific to this control. Type the properties for this control Keyword = Value format, for example, `MAX=10`. The available keywords and valid values are described by information associated with the extension. Click the **Extensions** tab in the Workspace palette and then double-click the **Slider Control** extension to view the available keywords and valid values.
Action	Displays the action or filter for the control. See Setting up actions on page 177 for more information.
Edit Action	Click this button to set the desired action or filter for the control. See Setting up actions on page 177 for more information.
Dimensions: Left	The left origin (in pixels) of the control. Type the desired value.
Dimensions: Top	The top origin (in pixels) of the control. Type the desired value.
Dimensions: Width	The width (in pixels) of the control. Type the desired value.
Dimensions: Height	The height (in pixels) of the control. Type the desired value.

Table 6.21 Slider SFX Custom control properties *(Continued)*

Property	Value
Target check box	The left-hand column displays a check box that indicates whether the property is identical for all application targets in the project. If the check box is checked, the property setting applies to all targets, and if it is not checked, the property setting applies to this target only. The target check box column only appears if the current project has more than one application target defined.

Configuring application properties

In this step, you set the project properties for your application. Application properties include the name of the application, which form appears when the application is opened, and which database formats to use.

1 Select **Edit > Properties** from the menu.

- The Project Properties dialog box appears, as shown in Figure 4.21 on page 61.

2 Edit the following project properties:

- **Name of Application:** type the name of your application.

  This is the name of the handheld application as it will appear to your end users.

- **Initial Form:** select the form you want to appear first in your application.

  The Initial Form setting determines which form appears first when the application runs on the handheld.

- **Creator ID**: choose a pre-registered SMS*n* ID, or specify your own unique four-character ID. You must register your unique Creator ID on the ACCESS Americas (formerly PalmSource, Inc.) website. This applies to PalmOS targets and Pocket PC targets that use the PalmDB (PDB) device database format (for complete device database portability between platforms). It is not used for Pocket PC targets that use the PocketPC DB (CDB) device database format.

  Use one of the ten Creator IDs that Satellite Forms has reserved ("SMS0" to "SMS9"), or enter your own unique Creator ID.

⚠ Caution Deployments to third parties should **always** have a unique ID to avoid potential conflicts with other existing applications. Deployments internal to an organization can take advantage of the reserved IDs, provided other applications within the organization are not already using them.

- **Version:** sets the version number of the application.

  Enter the major version of your application, and the minor version if necessary, for example 1 and 0.

- **Desktop DB Format:** specify the desktop database format your application will use.

- **Device DB Format:** specify the device database format your application will use.

Note Satellite Forms version 8 supports the Palm DB (PDB) database format only, and no longer supports the obsolete Pocket PC DB (CDB) format. The Palm DB format is used for both the Palm OS and Pocket PC platforms.

- **Down Key at Table End Creates Record**: determines behavior of the application when a user scrolls to the bottom of a table.

  Checked (default): a new record will be created at the end of the table when a user scrolls past the last record in the table.

  Unchecked: the handheld device will beep when a user reaches the end of the table, and no new record will be created.

- **Oracle Lite compatible tables:** formats tables for integration with Oracle Lite from Oracle Corporation.

  Checked: tables will be formatted for integration with Oracle Lite.

  Unchecked (default): tables are not formatted for integration with Oracle Lite.

- **Enable filter wildcard value**: determines if wild card values will be allowed for a filter and defines wild card values.

  Checked (default): wildcard value filter is enabled with the value entered.

  Unchecked: wildcard filter value is not enabled.

- **Create Launcher Application:** Check this box and select the desired icon file by clicking the ... button to generate a launcher application with the selected icon. The user can then simply tap the icon to launch the application.

Note This option to create the launcher application now applies to both the PalmOS and Pocket PC platforms (older versions of Satellite Forms handled Palm OS targets only using this method).

  - **B&W File:** Type the name of the black-and-white icon file for the launcher application in this box or click the ... button to browse for the desired icon file. All icon files must be Windows.BMP format graphics files. For Palm OS targets, the large icon size is 31x21pixels and the small icon size is 15x9 pixels. For Pocket PC targets, the large icon size is 32x32 pixels and the small icon size is 16x16 pixels.
  - **Backup:** Check this box to set the file properties so that Palm Hotsync makes a backup of the launcher application.
  - **Invisible:** Check this box to make the launcher application icon invisible on the handheld device.

The Project Properties dialog box should now contain the complete properties of your application, as shown in Figure 4.22 on page 62

3 Click **OK** to close the dialog box.

Creating and Assigning a launcher icon image for your application

Integrated Runtime

Starting with Satellite Forms Version 8, a new integrated runtime engine feature has been added to improve application deployment. This feature works by combining your application launcher icon with the Satellite Forms runtime engine executable, to create a customized runtime branded specifically for your application. Previously, the launcher icon app was a small “loader” customized with your icon that would load the standard RDK runtime engine to run your application. Now the launcher icon and runtime engine are integrated together into one.

The integrated runtime improves app deployment by having one less file to install as a part of your package. More importantly, it makes your deployed app more robust by reducing the chance that installing another Satellite Forms application might affect your installed application by overwriting the runtime engine with a different version. For the Palm OS platform, the RDK runtime engine and you icon are combined into your application PRC file. For the Windows Mobile/Pocket PC platform, the EXE runtime engine and your icon are combined into your application EXE file. The runtime engine DLL file is still required however, and is not integrated with the icon file.

In order to have MobileApp Designer generate an integrated runtime engine for your application, you must specify a proper launcher icon in the Project Properties.

Palm OS Satellite Forms allows you to specify color and small icons for your application icons. In the project's property dialog box, specify a black and white 31x21 pixel icon. It is not necessary to specify the color and small icons in order to compile and distribute your application, but missing icons will appear as "hearts" on the device. MobileApp Designer will use the suffix "-Small", "-Color", and "-Small-Color" to locate the other icon files. The suffix corresponds to the size and bit-depth of the icon file.

Starting with Satellite Forms 7, you can add high density icons for both the standard and small (list view) sizes of your application icon. High density icons look much sharper on the Palm application launcher screen, because they use four times as many pixels as the standard density icons. To add high density icons for your application, simply name your large color icon with the -Color-HD.bmp file suffix, and place it in the same folder as your -Color.bmp standard density icon bitmap file. To add an HD version of your small (list view) icon, simply name your icon bitmap with the -Small-Color-HD.bmp file suffix.

The HD versions of your icon bitmaps need to be exactly twice as wide and twice as tall as the standard density versions. Therefore, the high density version of your large size icon needs to be 62x42 pixels, compared to the standard density color icon bitmap size of 31x21 pixels. The high density small color icon needs to be 30x18 pixels, compared to the 15x9 size of the standard density version.

For example, if the project is associated with "MyIcon.bmp", MobileApp Designer will look for "MyIcon-Small.bmp", "MyIcon-Color.bmp", "MyIcon-Color-HD.bmp", “MyIcon-Small-Color.bmp”, and “MyIcon-Small-Color-HD.bmp” in the same directory as the black and white icon file.

Large icon files must be 31x21 pixels in size. Small icon files must be 15x9 pixels in size. Large high density icons must be 62x42 pixels, and small high density icons must be 30x18 pixels.

MobileApp Designer uses a simple color-matching algorithm to convert color BMP files into Palm icons, so your icon may look different on Palm devices. To eliminate this effect, create icons using Palm's color palettes only. To make this process easier, Satellite Forms provides you with template files that you can use when creating icons. You can find the files "Template-Small-Color.bmp", "Template-Small-Color-

HD.bmp", "Template-Color.bmp" and "Template-Color-HD.bmp" in the Templates directory within the Satellite Forms installation directory. Satellite Forms recommends using these template icon files as starter color icon files for your projects. Copy these files into your project directory, renaming them according the the file naming convention mentioned earlier, and edit them using Microsoft Paint. You can also use Microsoft Paint to create your icons. You can usually find this program in \Program Files\Accessories\

Note If you use Microsoft Paint, black and white icon files must be saved as Monochrome Bitmap; color icon files must be saved as 24-bit Bitmap.

Procedure Creating a color icon

1 Open one of the template files using MS Paint: Start > Programs > Satellite Forms 8.0 > Template.

2 Draw the icon within the box (any image within that boundary will be converted to the icon).

3 Transparency color determines the color that will be "invisible" on Palm. The default transparency color is bright green, so any pixels with this color would be transparent.

4 Use the "Pencil" tool to draw the icon.

5 To choose a different color, use the "Pick Color" tool and click to select the desired color from the palette.

6 When finished with the color icon, save it using the proper naming convention, e.g., "MyIcon-color.bmp".

7 Do not resize the template file.

Note You can copy and paste your black and white icon into the designated box.

Pocket PC

To have MobileApp Designer generate a custom launcher for your application using your own icon, there are two options. You can use a set of small and large color bitmap files, or you can use a multi-image Winodws ICO icon file. *The option to use bitmap files instead of an ICO file is new with Satellite Forms 8.*

To use multiple bitmap files, the large icon bitmap must be 32x32 pixels in size, and 16 or 256 colors. The small bitmap should be 16x16 pixels, 16 or 256 colors. The small icon needs to be named the same as the large icon, with a -Small filename suffix. When you specify the large icon filename, MobileApp Designer will look for the -Small file automatically in the same folder.

To use a multi-image Windows ICO icon file instead of bitmaps, follow the steps below:

Procedure Generating a custom launcher for Pocket PC application icons

1 Create a multi-image Windows Icon file (*.ICO) that contains one 16x16 and one 32x32 image. The 16x16 icon will be used when directory is viewed as listing. The 32x32 icon will be used for large icon view. [Note that Satellite Forms does not

include a tool to create Windows ICO files; you must use another tool to create the ICO files, or use a converter to convert BMP files into ICO files.]

2 Name the icon file Your ApplicationName.ico where Your ApplicationName is the name of your application project (***.sfa**) file. *Note that this may be different than the Name of Application in the project properties*.

3 Create an Images folder at the same directory level as your application source file (.SFA) and place the icon file into that folder. It is recommended that you place all your bitmap files and the icon files to the Images folder. Save the image in bitmap format to an Icon file in the Images directory.

Note See the sample in the Samples\Projects\Work Order in your Satellite Forms installed directory.

Creating a splash screen

Starting with Satellite Forms Version 8, a new feature has been added to display a splash screen when your application starts. This feature is optional, and if you do not use a splash screen the standard launch progress text from previous versions will be shown instead.

This feature was a late addition to Satellite Forms 8 and is not yet integrated into the Project Properties settings. In a future release the splash screen option will be enabled from the Project Properties.

To add a splash screen to your application, you need to place a properly named and formatted bitmap file into your {Project}\Images folder.

Palm OS The splash screen bitmap must meet these specifications:

- The image dimensions should not exceed 160x160 pixels. An image size of 160x160 pixels is recommended.
- The image should be saved as a 24bpp color BMP file. It will be converted by MobileApp Designer into the proper color format for the Palm OS platform.
- An icon bitmap file for the application must be specified in the Project Properties.
- The splash screen bitmap file must be named the same as the icon bitmap file, with a "**-splash.bmp**" filename suffix.
- The splash screen bitmap file must be placed into your {Project}\Images folder.

If you follow those specifications, the splash screen image will be included in your application and will be displayed when the app launches.

Pocket PC The splash screen bitmap must meet these specifications:

- The image dimensions should not exceed 240x320 pixels. An image size of 240x320 pixels is recommended.
- The image should be saved as a 8bpp color BMP file.
- An icon bitmap file for the application must be specified in the Project Properties.
- The splash screen bitmap file must be named the same as the icon bitmap file, with a "**-splash.bmp**" filename suffix.

- The splash screen bitmap file must be placed into your {Project}\Images folder.

If you follow those specifications, the splash screen image will be included in your application and will be displayed when the app launches.

The Colorizer sample project demonstrates the use of a splash screen bitmap, as well as the integrated runtime engine and color forms & controls. If you installed Satellite Forms in the default location, this sample is located at C:\Satellite Forms 8\Samples\Projects\Colorizer\Colorizer.sfa and can also be reached from the Windows Start menu > Programs > Satellite Forms 8 > Samples > Projects > Colorizer. The Colorizer sample project has also been precompiled into a Redistribution sample that is ready to install onto a Palm OS or Windows Mobile handheld, via the Windows Start menu > Programs > Satellite Forms 8 > Samples > Redist > Colorizer.

Installing the engine and downloading the application

Complete the following steps to install your Satellite Forms Palm and Pocket PC applications to your handheld device for testing.

Install the engine on Palm device

Procedure Install the Satellite Forms runtime engine/Palm SDK engine

1. Click Start and point to Programs > Satellite Forms 8.0 > Palm > Runtime and click **Install SDK Engine**.
2. After the installation begins, select the username of the device on which you want to install the SDK engine and click **Install**. Follow the prompt to perform a Hotsync that installs the runtime engine.

Install the engine on Pocket PC device

Procedure Install the Satellite Forms runtime engine/Pocket PC SDK engine

Note If you use the new Satellite Forms 8 feature to generate an integrated runtime engine for your Pocket PC application (described in the previous section), the engine install step can be omitted. You can skip ahead to the Download step below. The integrated runtime engine will be installed with your application.

- Click Start and point to Programs > Satellite Forms 8.0 > Pocket PC and click SDK Runtime Installer. Follow the prompts to complete the installation.

Download the application

Procedure Download the application onto the device

1. Place the handheld device into its cradle.
2. Select Download App & Tables from MobileApp Designer Handheld menu or click the Download App & Tables shortcut icon.
3. Make sure ActiveSync or Hotsync Manager is running, then initiate a synchronization from the button on the cradle to transfer the application to the device.

Testing the application

After you have downloaded your application to the handheld device, you should perform complete testing of the application to make sure it performs as designed. Testing procedures will vary based on the type of application you have created. You can get started with testing using the steps below:

Procedure Test the sample application

1 For Palm applications, tap the **Applications** button on the handheld's screen with your stylus. For Pocket PC platforms, selet Start > Programs. Then tap the **Sat Forms** icon to start Satellite Forms MobileApp Designer.

2 From the **Select Application to Run** list, tap on the name of your application.The form you set as the Main form should appear. Verify that the appropriate records or information is displayed.

3 Test each piece of your application, opening each form, and verifying that the complete application works as designed.

4 When you are finished, from the Options menu select Exit to close the application.

The final step of testing the application should include uploading the application to MobileApp Designer to make sure new updates and data entries are being handled correctly. For instructions on uploading application tables from the handheld device to MobileApp Designer, see Installing the engine and downloading the application on page 175.

The next step

Phase 1 of creating your Satellite Forms application is now complete. The next phase is covered inChapter 8.

Chapter 7
Using Actions, Filters, Extensions, and Color

This chapter explains how to use control actions and filters to enhance the functionality of your Satellite Forms applications. It covers setting control actions and filters in detail and introduces SFX plug-ins, which are covered in detail in Satellite Forms API Reference, on page 467. The capabilities (*new in Satellite Forms version 8*) for using color forms and controls to enhance the appearance of your application are also explored.

Setting up actions

Where appropriate, MobileApp Designer controls and all Menu Items support a property called **Action** that occurs when a user taps the control or selects the Menu Item. The Action property allows you to specify how the control affects the application, for example, jumping to a different form or setting a table filter.

Note A control always carries out its primary function before the operation specified in the Action property occurs. For example, if you set up a Check Box control to jump to another form when it is checked or unchecked, the control performs its primary function, placing a T or F in its data source column, before jumping to the other form.

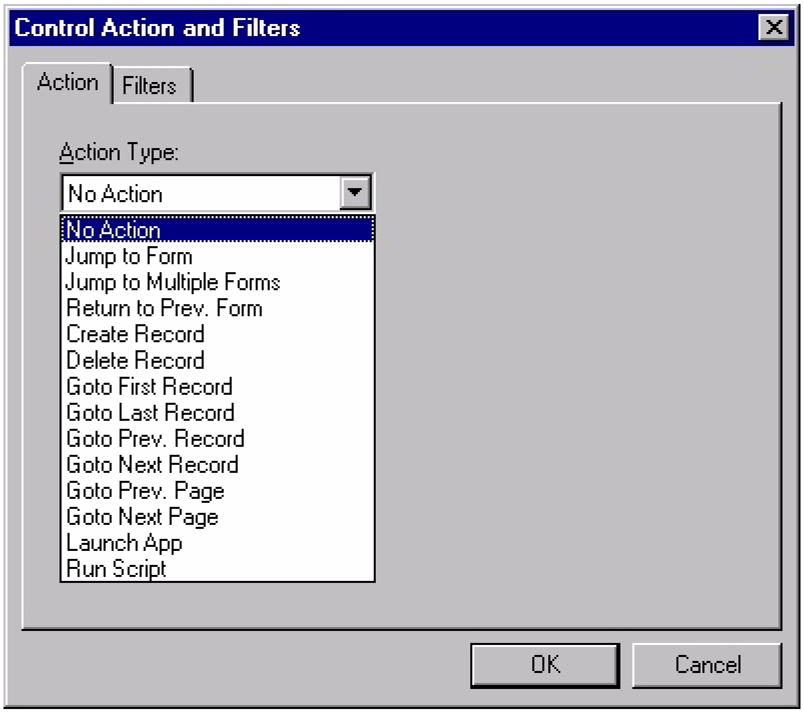
The controls that include the Action property are:

- Button control
- Check Box control
- Drop List control
- List Box control
- Radio Button control
- Many SFX Custom controls
- Menu Items other than Separators or Edit menus

Control actions

The Propertyspace palette for each item listed above includes the **Action** property. Double-click the **Action** value or click the **Edit Action** button to open the **Control Action and Filters** dialog box, as shown in the following figure:

Figure 7.1 Control Action and Filters dialog box



Display the actions available for a specific control by clicking the arrow on the **Action Type** combo box, as shown above.

The following table lists all control actions. Not all actions are available for every control.

Table 7.1 Control actions

Action Name	Description
No Action	No action is defined.
Jump to Form	The application displays the specified form. If the target form is linked to the same table as the current form, the target form display the same record as the current form. If the target form is linked to a different table, the target form displays the first record of the new table. See Jump to Form options on page 179 for detailed information on the available options for this action.
Jump to Multiple Forms	The application displays a target form based on the contents or state of a specified control. Allows users to navigate automatically to different forms based on the data they enter in the source form. See Jump to Multiple Forms options on page 180 for detailed information on the available options for this action.

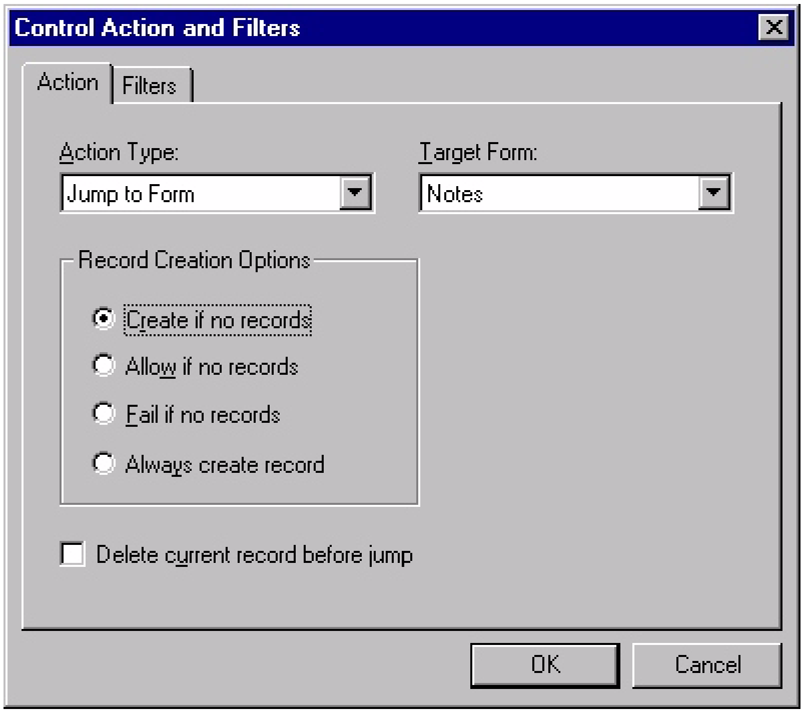
Table 7.1 Control actions *(Continued)*

Action Name	**Description**
Return to Prev. Form	The application returns to the previously displayed form. • **Delete current record before jump:** Deletes the current record on the current form before the jump occurs. Typically used to cancel the addition of a new record to a table linked to a secondary form.
Back to Previous	The application accesses 5.2.2 forms stored in a list.
Create Record	Creates a new record in the current form’s linked table.
Delete Record	Deletes the current record from the form’s linked table.
Goto First Record	Displays the first record in the form’s linked table.
Goto Last Record	Displays the last record in the form’s linked table.
Goto Prev. Record	Displays the previous record in the form’s linked table.
Goto Next Record	Displays the next record in the form’s linked table.
Goto Prev. Page	Displays the previous page on a multiple-page form. If the current page is the first page, this action displays the last page on the form.
Goto Next Page	Displays the next page on a multiple-page form. If the current page is the last page, this action displays the first page on the form.
Launch App	Closes the current application and runs the specified application. See Launch App options on page 183 for detailed information on the available options for this action.
Run Script	Executes the selected script. See Using the Run Script action on page 184 for detailed information on the available options for this action.

Jump to Form options

When you select the **Jump to Form** action type, the Control Actions and Filters dialog box appears as show in the following figure:

Figure 7.2 Control Actions and Filters dialog box, Jump to Form action



Action Type: Jump to Form

Target Form: Notes

Record Creation Options

● Create if no records

○ Allow if no records

○ Fail if no records

○ Always create record

□ Delete current record before jump

The available options are:

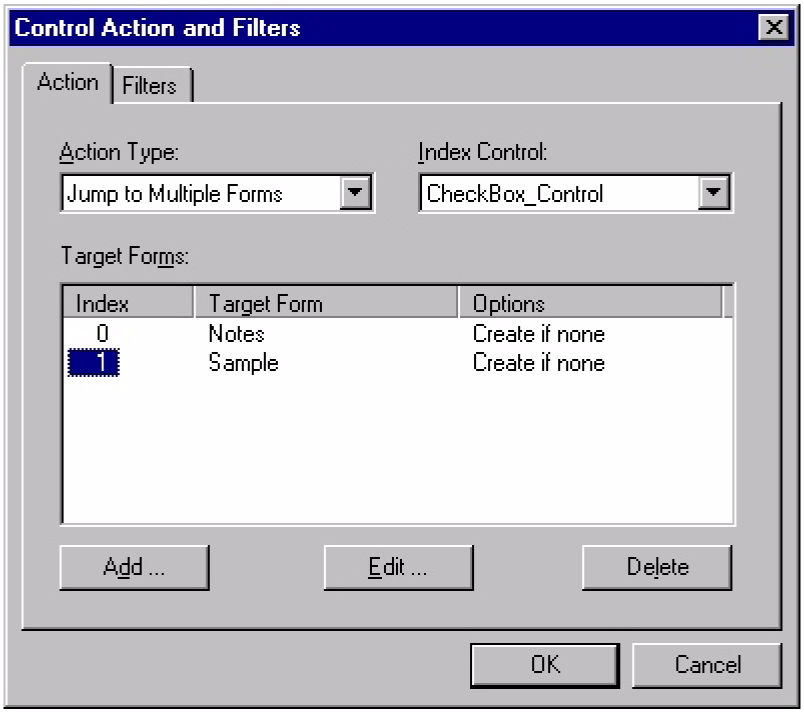
- **Target Form:** Select the form that is the target of the jump.
- **Create if no records:** If the target form's linked table is empty, creates a new record when the jump occurs.
- **Allow if no records:** If the target form's linked table is empty, no new record is created when the jump occurs.
- **Fail if no records:** If the target form's linked table is empty, the jump fails. The handheld device displays a dialog box with the message No records exist.
- **Always create record:** Creates a new record in the target form's linked table when the jump occurs.
- **Delete current record before jump:** Deletes the current record on the current form before the jump occurs. Typically used to cancel the addition of a new record to a table linked to a secondary form.

Jump to Multiple Forms options

The **Jump to Multiple Forms** action makes it possible to jump to different forms based on the value in a specified control. In an application, you can build different forms for different purposes and then jump to the correct form based on the setting, for example, of a Check Box control or a group of Radio Button controls.

When you select the **Jump to Multiple Forms** action type, the Control Actions and Filters dialog box appears as show in the following figure:

Figure 7.3 Control Actions and Filters dialog box, Jump to Multiple Forms action



Control Action and Filters

Action Type: Jump to Multiple Forms

Index Control: CheckBox_Control

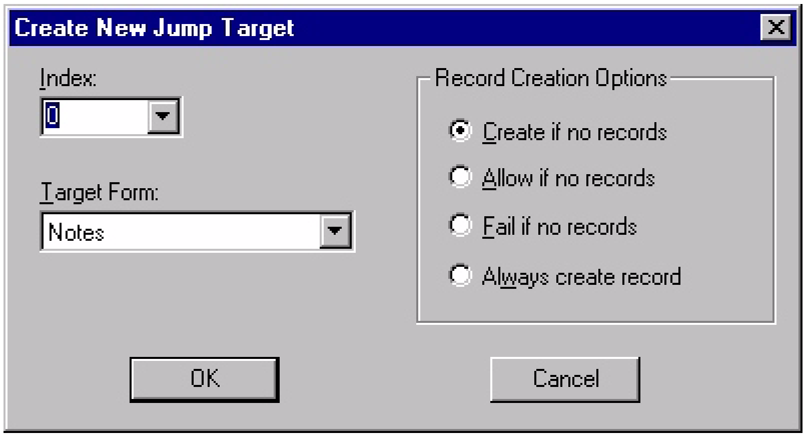
Target Forms:

Index	Target Form	Options
0	Notes	Create if none
1	Sample	Create if none

The available options are:

- **Index Control:** The control on the current form whose setting determines which form the jump accesses. Select the desired control from the combo box.
- **Target Forms:** Lists the target forms defined for the multiple jump, including the control index value, target form, and jump options for each defined jump.
- **Add... Button:** Click this button to add a new jump to the Target Forms list. The Create New Jump Target dialog box appears as shown in the following figure:

Figure 7.4 Create New Jump Target dialog box



Create New Jump Target

Index: 0

Target Form: Notes

Record Creation Options

- ● Create if no records
- ○ Allow if no records
- ○ Fail if no records
- ○ Always create record

The options on the Create New Jump Target dialog box are:

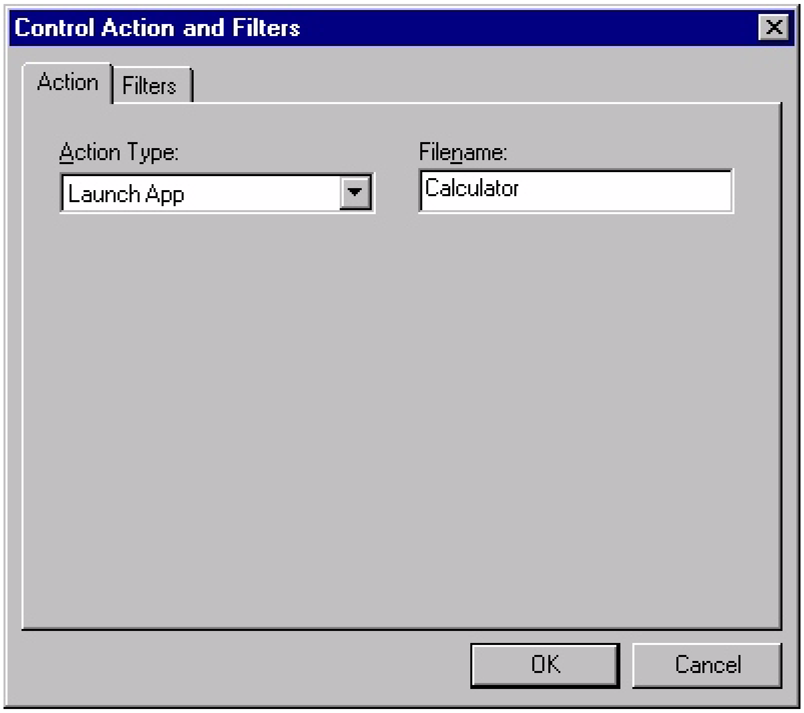
- **Index:** Select the desired index for this jump from the combo box.

  - **Target Form:** Select the target of the jump from the combo box. When the control index value is equal to the **Index** setting, the application jumps to the selected form.
  - **Record Creation Options:** See Jump to Form options on page 179 for more information on these options.
- **Edit... Button:** Click this button to edit the jump selected in the Target Forms list. The Edit Jump Target dialog box offers exactly the same options as the Create New Jump Target dialog box, as shown in Figure 7.4.
- **Delete:** Click this button to delete the jump selected in the Target Forms list. MobileApp Designer deletes the selected jump immediately, without prompting you first.

Launch App options

This action closes the current application and runs the specified application. When you select the **Launch App** action type, the Control Actions and Filters dialog box appears as show in the following figure:

Figure 7.5 Control Action and Filters dialog box, Launch App action



Action Type: Launch App

Filename: Calculator

The names of some the built-in handheld applications on Palm OS devices are listed in the following table. To launch a Palm OS application, type the application file name without the .PRC extension. To launch a PocketPC application, enter the full path and filename with the .exe suffix, eg. \Windows\Calc.exe.

Table 7.2 Built-in application names

Application Picker Label	File Name to Enter
Address	Address Book
Calc	Calculator
Date Book	Date Book
Expense	Expense
HotSync	HotSync
Mail	Mail
Memo Pad	Memo Pad
Prefs	Preferences
Security	Security
To Do List	To Do List

Note that in addition to the **Launch App** control action type, there is a related function in the SysUtils extension named SU_LaunchApp that allows you to specify the app name at runtime. This offers some flexibility over the **Launch App** control action which requires that you specify the app name at design time.

Using the Run Script action

This action runs the script associated with the control. Scripts provide access to calculations, conditional logic, bounds checking, and Satellite Forms extensions, both SFX plug-ins and controls, written with the Satellite Forms API. For more detailed information on the scripting language and its syntax, see Satellite Forms Scripting Language Reference, on page 257.

When you select the **Run Script** action type, Control Action and Filters dialog box has just one option: the **Edit Script...** button. Click this button to open the script editing window for the current control. Type the desired script code and save and build the project to test the new script.

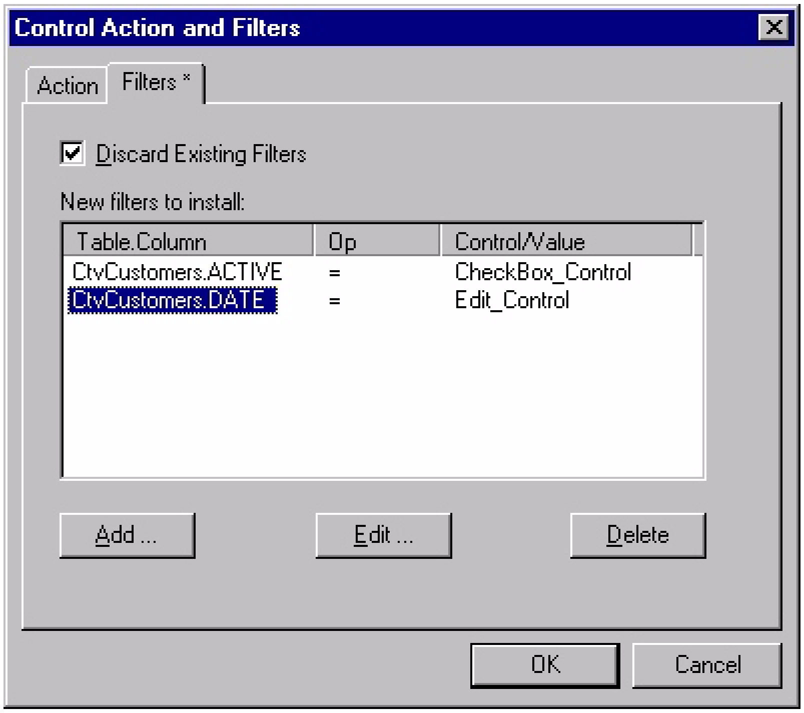
Using table filters

All controls and Menu Items that support actions can also apply filters to the tables in an application. A table filter uses a criterion, which is simply a mathematical operator, and a value to select a subset of records in a table for use at a given stage in an application. The value can be fixed, a wildcard, or a snapshot of a specific control at the time the filter is applied. Records that do not meet the filter criterion are not available until you remove the filter.

Filters and actions operate independently of each other. A control can implement an action, a filter, or both. If you apply both an action and a filter to a control, you generally design them to work together, for example, jumping to a form and displaying a filtered subset of data.

The Control Action and Filters dialog box, with the Filters tab open and two sample filters applied, appears as shown in the following figure:

Figure 7.6 Control Action and Filters dialog box, Filters tab



☑ Discard Existing Filters

New filters to install:

Table.Column	Op	Control/Value
CtvCustomers.ACTIVE	=	CheckBox_Control
CtvCustomers.DATE	=	Edit_Control

In the example shown above, the first filter displays records from the CtvCustomers table only if the value of the ACTIVE field is equal to the current value of CheckBox_Control. If CheckBox_Control is checked, its value is T. Using the example filter, only records from the CtvCustomers table in which the ACTIVE field equals T are available.

The second filter compares the value of Edit_Control with the value of the DATE field. Only records from the CtvCustomers table that pass through the first filter are subject to the second filter. Within that filtered subset of records, only those in which the DATE field equals the value entered in Edit_Control are available.

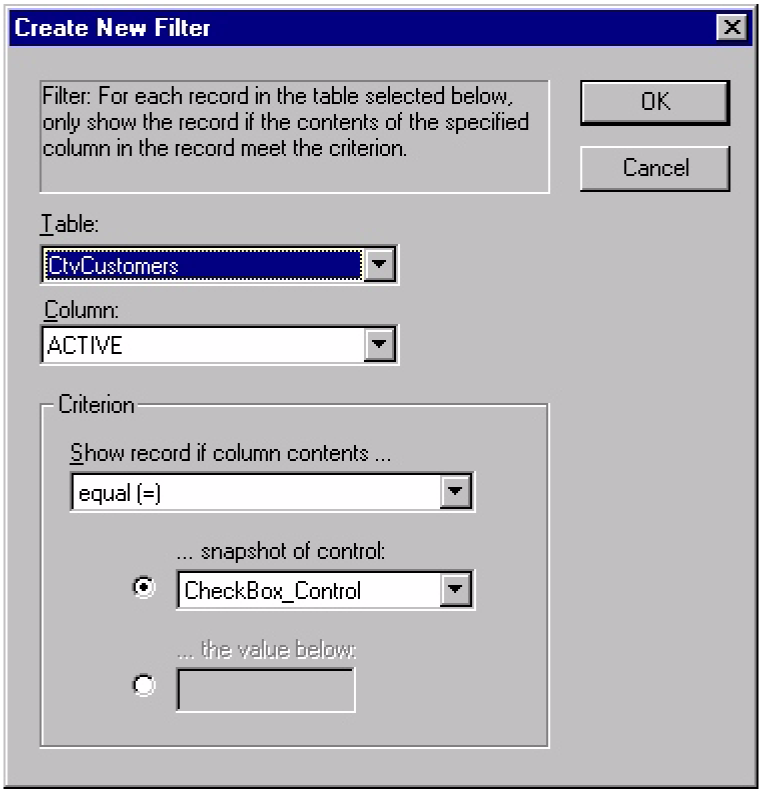
Using multiple filters allows you to define a highly specific set of records for use at a given point in an application. Filters make your applications more efficient and easier to use by presenting only the data a user needs to complete a specific task. With proper filtering, users do not need to browse through dozens of records to find the one they need.

Note A control always executes its primary function before the applying a filter. For example, if you set up a Check Box control to apply a filter when it is checked or unchecked, the control performs its primary function, placing a T or F in its data source column, before applying the filter.

Adding or editing a filter

To add a new filter, click the **Add...** button on the Filters tab of the Control Action and Filters dialog box. To edit an existing filter, click the desired filter in the list on the Filters tab of the Control Action and Filters dialog box and then click the **Edit...** button. Clicking either button opens a dialog box with the options available on the Create New Filter dialog box, as shown in the following figure:

Figure 7.7 Create New Filter dialog box



Filter: For each record in the table selected below, only show the record if the contents of the specified column in the record meet the criterion.

Table: CtyCustomers

Column: ACTIVE

Show record if column contents ... equal (=)

● ... snapshot of control: CheckBox_Control

○ ... the value below:

The Create New Filter/Edit Filter dialog box contains the following options:

- **Table:** Select the table from the combo box to which to apply this filter. Only tables that are in the current project are available.
- **Column:** Select the column in the selected table on which to filter from the combo box.
- **Criterion: Show record if column contents...:** Select the desired mathematical operator to be used when comparing the value of the selected column to either the control snapshot or value. See Mathematical operators for filters for more information.
- **Criterion: ...snapshot of control:** Click the associated radio button and select the control whose value at the time the application applies the filter is to be the basis of the filter from the combo box. Only controls in the current project are available.
- **Criterion: ...the value below:** Click the associated radio button and type the actual value or wildcard that is the basis for the filter in the edit box.

The valid wildcard characters are defined in the **Project Properties** dialog box, as shown in Figure 5.13 on page 83, using the **Enable Filter Wildcard Value** check box and edit box.

Mathematical operators for filters

All filters require a mathematical operator to determine how the filter works, regardless of the value the filter uses. The available operators are:

- **Equal (=):** The two operands – the table record and the comparison value – must be identical.
- **Not Equal (< >):** The two operands – the table record and the comparison value – must be different.
- **Less Than (<):** The first operand – the table record – must be less than the second operand.
- **Less Than or Equal (< =):** The first operand – the table record – must be less than or equal to the second operand.
- **Greater Than (>):** The first operand – the table record – must be greater than the second operand.
- **Greater Than or Equal (= >):** The first operand – the table record – must be greater than or equal to the second operand.
- **Equal Anything (del filter):** The table record is not filtered. Applying this criterion removes filtering from the selected table.
- **Contain:** The first operand – the table record – must contain the comparison value, whether at the beginning, middle, or end. The location in which the comparison value appears in the table record is unspecified.
- **Begin With:** The first operand – the table record – must begin with the comparison value.

Note Date, Time, and Numeric fields do not support filtering based on the Contain and Begins With filter operators.

Filter tips

When you use filters in your Satellite Forms applications, bear in mind the following conditions:

- Filters apply to the entire application. When you apply one or more filters to a table, it is the table that is filtered. Therefore, all controls – not just the one that you used to apply the filter – linked to that table can retrieve only the records that meet the filter criteria.
- When you apply a filter to a table, the filtered table behaves as if it had been converted to a new table containing only the records that meet the filter criteria.
- Selecting **...snapshot of control** as the filter criterion means the filter uses the contents of the control at the time the application applies the filter. To apply a different filter value, you must first remove the current filter.
- If you create a new record in a filtered table, the new record is created with fields initialized to meet the filter criteria.
- If you apply a filter to a table using criteria based on a field that is the basis the current filter, the new filter criteria for that one field overwrite the previous filter. The new filter does not affect filters based on other fields of the same table.
- When you apply a filter to the current form's linked table, the form fires an `OnValidate` event because it needs to reload the table data with the filter applied. See OnValidate on page 415 for more information.
- Filters may also be managed from scripts, with the AddFilter, RemoveFilter, and RemoveAllFilters functions.

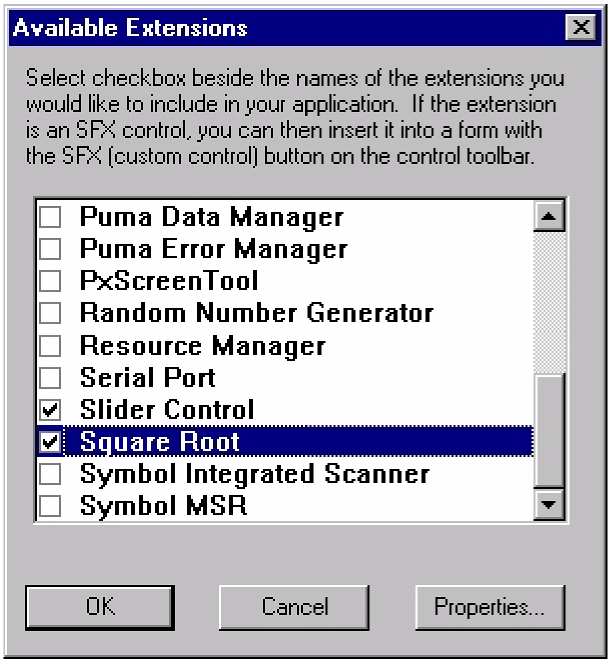
Adding extensions to Satellite Forms

The Satellite Forms Application Programming Interface (API) lets you extend the capabilities of Satellite Forms applications with extensions called SFX plug-ins and SFX controls. You can write your own extensions or use extensions others have written. Satellite Forms provides several extensions, including an SFX plug-in called Square Root and an SFX control called Slider.

This section uses the example of the Square Root SFX plug-in to illustrate how to incorporate an SFX Extension into an application.

To add an SFX Extension to a Satellite Forms application, select **View** > **Extensions** from MobileApp Designer menu or click the **Manage Extensions** button on the Misc Toolbar. When the **Available Extensions** dialog box appears, as shown in the following figure, scroll to **Square Root** and check box to the left of the Extension name.

Figure 7.8 Available Extensions dialog box



Available Extensions

Select checkbox beside the names of the extensions you would like to include in your application. If the extension is an SFX control, you can then insert it into a form with the SFX (custom control) button on the control toolbar.

- ☐ Puma Data Manager
- ☐ Puma Error Manager
- ☐ PxScreenTool
- ☐ Random Number Generator
- ☐ Resource Manager
- ☐ Serial Port
- ☑ Slider Control
- ☑ Square Root
- ☐ Symbol Integrated Scanner
- ☐ Symbol MSR

All Extensions added to the current project are listed in the Workspace palette. Click the plus sign next to the Extension name to display its methods. Double-click a method name to display a dialog box containing information about the method. For example, the Square Root SFX plug-in has two methods: About and SqrRoot. The About method displays up the extension's **About** dialog box, providing general information. The SqrRoot method performs the square root calculation.

Note When you add an extension to a project, MobileApp Designer automatically enables the **Handheld > Include Extensions in Download** option. The next synchronization downloads all extensions to the handheld device when this option is enabled. It is unnecessary to download the extension again unless it has changed. As a result, MobileApp Designer turns off the option after the application download is completed. It enables the option again if you add another extension. Every time you open a project that includes extensions, MobileApp Designer enables this option until the first download completes.

You can now use the Square Root plug-in in any scripts you write. In the examples below, an Edit control named OutputA, receives the square root of the value in the Edit control named InputA. The SqrRoot method of the Square Root extension is a global method, which allows you to reference it directly by name. For example, the following script is valid:

```
'Example of using Square Root plug-in with the scripting language
'Output and InputA are edit controls.
'Perform the Square Root calculation.
OutputA = SqrRoot(InputA)
```

You can always reference an extension by its full name, for added clarity, in the form: `Extensions().Method()`, as shown in the following example:

```
'Second example of using Square Root plug-in with the scripting language
'Output and InputA are edit controls.
'Display the dialog box.
```

```
Extensions("SqrRoot").About
'Perform Square Root calculation.
OutputA = Extensions("SqrRoot").SqrRoot (InputA)
```

For information on and examples of how to use the API to create your own SFX plug-ins, see Creating an SFX plug-In, on page 468.

Using color in your application

Satellite Forms version 8 introduces a new capability to enhance your application with color forms and controls, using the Colorizer extension. Colorizer provides script functions to change colors of various user interface (UI) elements, enabling you to enhance the visual appeal of your application, or to convey information with more impact. You can set a color theme to use throughout your application, modify colors on a form-by-form basis, or even change colors while a form is displayed. The Colorizer extension works on both the Windows Mobile and Palm OS platforms.

In a broad sense, the Colorizer extension works by allowing you to set foreground and/or background colors for object *types* rather than for individual objects. This means that when you set a color for an edit control, for example, that color is set for all edit controls rather than for just a single specific control. This approach is perhaps more easily understood by thinking of it as color *themes* for your application, rather than a way to uniquely colorize individual controls. A benefit of this approach is that by working on an *object type* basis rather than *per-object* basis, a consistent visual appearance throughout the application is more easily achieved.

In order to use Colorizer in your application, you need to add the Colorizer extension to your project, via the Manage Extensions toolbar icon. Click on Manage Extensions, then select Colorizer from the list of available extensions. The Colorizer script functions are then available to your application. See Adding Extensions to Satellite Forms for a more detailed description of this step, if needed.

While the Colorizer extension is provided for both handheld platforms, there are some important differences that relate to how each platform handles color objects. It's important to be aware of these differences if your application is targeting both platforms, as you may need to adjust your scripting based on the platform. These differences will be explained in detail below.

In general, the Windows Mobile platform allows more differentiation between the types of controls than the Palm OS platform does. So, while you can apply different colors for buttons, checkboxes, radio buttons, and droplists on Windows Mobile, those controls are all treated the same on Palm OS. It may sound restrictive at first, but even given the limitations, you can add significant visual appeal to your Palm OS applications using Colorizer.

Platform-specific considerations

- Windows Mobile applications can be customized more so than Palm OS, but have the added requirement that you either need to define colors for all object types, or none at all.
- For Windows Mobile, you need to specify all of the object type color values, then enable color support with the `Colorize(true)` method.

- For Windows Mobile, you can revert to the system default colors simply by calling `Colorize(false)`.
- For the Palm OS platform, color changes are effective immediately without needing to call the `Colorize` method. The `Colorize` method is simply ignored on Palm OS.
- For the Palm OS platform, there is no way to easily revert back to the system default colors. You would need to know what the default color values were and set the current object colors back to those values.
- For the Palm OS platform, there are some additional uncommon UI object types that can be colored using `ColorizeExtra`, which is ignored on the Windows Mobile platform.

Color Values

All of the functions in Colorizer accept color values in a specific format, as hexadecimal RGB values in bbggrr (blue green red) order. The red, green, and blue values are from a range of 0..255, or 00..FF in hex format. The RGB color value should be prefaced with the **&H** operator to denote that it is a hexadecimal value. An example value for a solid, full intensity blue color would be `&HFF0000`. The blue intensity level is 255 (FF), while green and red are both zero. Similarly, solid bright green would be `&H00FF00`, and bright red would be `&H0000FF`. Solid black is `&H000000`, while solid white is `&HFFFFFF`. Bright yellow is a mix of green and red (`&H00FFFF`), magenta is blue & red (`&HFF00FF`), and cyan is blue & green (`&HFFFF00`). Medium orange would be something like `&H1386FB`, and bright pink might be `&H7513FB`, and so on. Various shades of grey have the same intensity values for blue, green, and red, so a medium grey might be `&H808080` for example. This RGB format for specifying colors is probably familiar to web page designers, as a similar approach is used in the HTML web page markup language.

Tip Many paint programs on PCs have a color selection tool that will help you derive the hexadecimal RGB values for any color you choose.

Coloring the Form

The form is probably the first place to start when adding color to your application. The Colorizer function `ColorizeForm(backcolor)` is used to set the background color of the form. To set the form background color to bright blue, for example, you would call `ColorizeForm(&HFF0000)`.

As mentioned above, for the Windows Mobile platform, you need to set all object type colors, and then enable those colors by calling `Colorize(true)`. On the Palm OS platform, the form color is automatically used as the background color for many of the control types, so when you call `ColorizeForm`, it also affects the background color of most controls.

Coloring Controls

- Button controls can be colored using the `ColorizeButton(forecolor, backcolor)` method.

- Text controls can be colored using the `ColorizeText(forecolor, backcolor)` method.
- Checkbox controls can be colored using the `ColorizeCheckbox(forecolor, backcolor)` method.
- Radio button controls can be colored using the `ColorizeRadio(forecolor, backcolor)` method.
- Edit controls can be colored using the `ColorizeEdit(forecolor, backcolor)` method.
- Paragraph controls can be colored using the `ColorizeParagraph(forecolor, backcolor)` method.
- Droplist controls can be colored using the `ColorizeDroplist(forecolor, backcolor)` method.
- List controls can be colored using the `ColorizeListbox(forecolor, backcolor)` method.
- Lookup controls can be colored using the `ColorizeLookup(forecolor, backcolor)` method.
- Ink controls can be colored using the `ColorizeDroplist(forecolor, backcolor)` method.

Setting Extra Colors

On the Palm OS platform only, there are some additional color settings that can be accessed using the `ColorizeExtra(UIelement, color)` script method. In fact, it is possible to set any of the form, control, and extra object colors using the `ColorizeExtra` method. The UI element is specified via an index number. The list of user interface elements that can be colorized with this function, and their index numbers, is:

Table 7.3 Palm OS user interface element types

Index	Element Name	Description
0	UIObjectFrame	Color for the border of user interface objects (such as command buttons, push buttons, selector triggers, menus, arrows checkboxes, and other controls).
1	UIObjectFill	The background color for a solid or "filled" user interface object.
2	UIObjectForeground	The color for foreground items (such as labels or graphics) in a user interface object.
3	UIObjectSelectedFill	The background color of the currently selected user interface object, whether that object is solid or not.
4	UIObjectSelectedForeground	The color of foreground items in a selected user interface object.
5	UIMenuFrame	The color of the border around the menu.

Table 7.3 Palm OS user interface element types

Index	Element Name	Description
6	UIMenuFill	The background color of a menu item.
7	UIMenuForeground	The color of the menu's text.
8	UIMenuSelectedFill	The background color of a selected menu item.
9	UIMenuSelectedForeground	The color of the text of a selected menu item.
10	UIFieldBackground	The background color of an editable text field.
11	UIFieldText	The color of the text in the editable field.
12	UIFieldTextLines	The color of underlines in an editable field.
13	UIFieldCaret	The color of the cursor in an editable text field.
14	UIFieldTextHighlightBackground	The background color for selected text in an editable text field.
15	UIFieldTextHighlightForeground	The color of the selected text in an editable text field.
16	UIFieldFepRawText	Special setting used only on Japanese devices.
17	UIFieldFepRawBackground	Special setting used only on Japanese devices.
18	UIFieldFepConvertedText	Special setting used only on Japanese devices.
19	UIFieldFepConvertedBackground	Special setting used only on Japanese devices.
20	UIFieldFepUnderline	The color used for underlines in the inline conversion area.
21	UIFormFrame	The color of the border and titlebar on a form.
22	UIFormFill	The background color of a form.
23	UIDialogFrame	The color of a border and titlebar on a modal form.
24	UIDialogFill	The background color of a modal form.
25	UIAlertFrame	The color of the border and titlebar on an alert panel.
26	UIAlertFill	The background color of an alert panel.

Chapter 8
Integrating with your Database

The chapter provides an overview and description of the process of integrating Satellite Forms applications with your desktop Database Management System (DBMS). These instructions assume you are familiar with the terms and concepts relating to DBMSs.

Overview

MobileApp Designer allows you to create applications consisting of forms and data tables and then download those applications to handheld devices. In some cases, this is the end of the development process. If you are using the data you collect exclusively on the handheld device and do not need to copy it to a desktop computer to save it or merge it with an existing database, you need not do anything further.

In most cases, however, you need to extract the data your Satellite Forms applications display from a database in a DBMS on a desktop computer or server and merge information collected or modified using the handheld device back into the same database. The Satellite Forms HotSync and ActiveSync ActiveX controls make this a simple process. The Satellite Forms HotSync and ActiveSync controls work with any DBMS that supports 32-bit ActiveX controls, also called OLE controls or custom controls.

Most popular DBMS products, including Microsoft Access, Lotus Approach, and so on, support ActiveX controls. Satellite Forms also supports a DLL-based integration method for DBMS products that do not support ActiveX controls.

Integrating your application with your database management system is Phase 2 of the application development process. Phase 1 is covered in Phase 1: Working with MobileApp Designer, on page 103. To integrate your application, you must complete the following steps:

1. Link the database tables created by Satellite Forms with the database application.
2. Write code to extract data from the database and copy it into the Satellite Forms tables. This code prepares a table or tables to be downloaded to the handheld device.
3. Write code to transfer data from the associated Satellite Forms database tables on the handheld device back to the database and merge as desired.

4 Write code to handle sync events, that is, to transfer information to and from the handheld device. This code calls the code in Steps 2 and 3 as required.

Overview of the integration process

MobileApp Designer saves the data tables Satellite Forms applications use in Access .MDB or dBase .DBF format. The Access and dBase file formats are widely used as a universal information interchange medium. A variety of desktop DBMS products, organizers, spreadsheets, and so on, support these database formats.

The following steps summarize the process of integrating a Satellite Forms application with a DBMS:

Procedure Integrating a Satellite Forms application with a DBMS

1 Extract the desired information from the database and copy it into the tables MobileApp Designer creates.

2 Download the tables to the handheld. The user in the field views and edits the information in these tables.

3 Copy the tables on the handheld into the tables MobileApp Designer created. These tables now contain the changes made by the user.

4 Inspect and merge, if desired, the data in the tables into your database. The merging procedure is entirely under the control of your DBMS or custom desktop application.

The actual details of how to transfer information from the tables in your DBMS to and from the tables MobileApp Designer created for your Satellite Forms application depends on your DBMS product and is not covered in detail here. Virtually all DBMS products can either link to Access or dBase tables using ODBC or import and export Access or dBase tables.

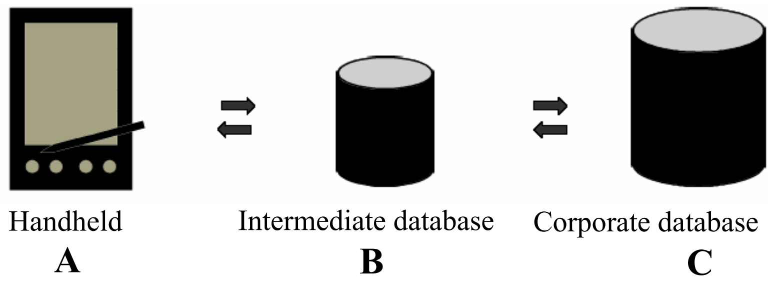
Integrating a Satellite Forms database with a Corporate database

Satellite Forms applications that interchange data with a Corporate database store data in three locations:

- the handheld
- the Satellite Forms intermediate databases (either DBF or MDB)
- the Corporate database

The data storage architecture is illustrated below.

Figure 8.1 Satellite Forms data storage architecture



As the arrows indicate, data must pass through the intermediate data transfer tables (B) to be moved between the Handheld and the Corporate Database. Some or all of the data exchanges indicated by the arrows will occur during each sync (HotSync or ActiveSync) operation. The developer can control which data exchanges occur and the order in which they occur. This gives complete flexibility in managing the sync process. Note that the data transfer tables are used as temporary storage areas to facilitate the data transfer. Data in these tables are completely overwritten by successive sync operations.

To successfully integrate Corporate Data with your Satellite Forms application, you must understand how to accomplish four data transfers (refer to the previous figure) described as follows:

- A -> B
- B -> C
- C -> B
- B -> A

You must also understand how to manage the sync operation using the methods and events of the Satellite Forms HotSync and/or ActiveSync ActiveX Control. In the example below, we describe each of these steps in detail.

Note Direct synchronization of Satellite Forms handheld data with server databases (A <--> C) is possible with third party server synchronization products. To find out more about these options, consult the Satellite Forms Solutions Guide in the Satellite Forms docs folder or online at http://www.satelliteforms.net/solutions.htm.

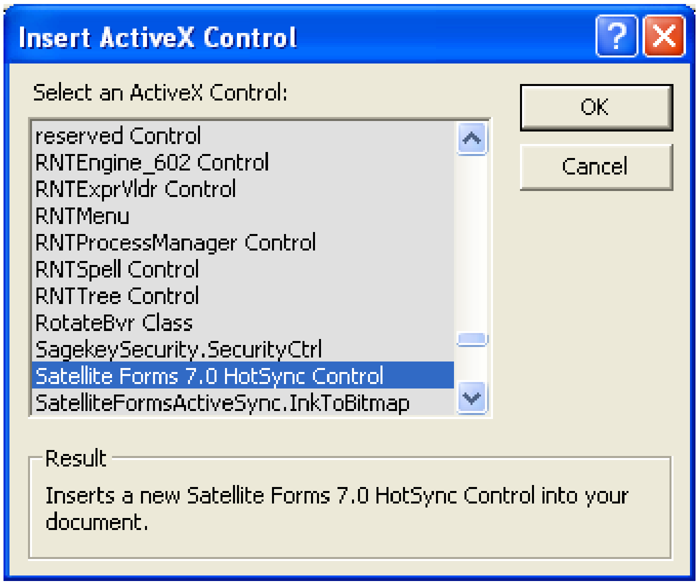
Satellite Forms HotSync ActiveX control for Palm OS

The Satellite Forms HotSync ActiveX control allows you to interact with the HotSync process of the Palm OS handheld, making it possible for you to transfer Satellite Forms tables and applications between desktop computers and handheld devices.

The Satellite Forms HotSync ActiveX control is installed and registered on your PC automatically during installation. To use the ActiveX control with your DBMS, you need to place the control on a form in a database application. The following example uses Access 2000.

To place the Satellite Forms HotSync ActiveX control onto an Access 2000 form, open the desired database, click the **Objects > Forms** button on the left side of the database window, click the desired form in the list, then click the **Design** button on the database window toolbar. Then select **Insert > ActiveX Control...** from the Access 2000 menu and scroll to and click **Satellite Forms 8.0 HotSync Control**, as shown in the following figure:

Figure 8.2 Access 2000 Insert ActiveX Control dialog box



Select an ActiveX Control:

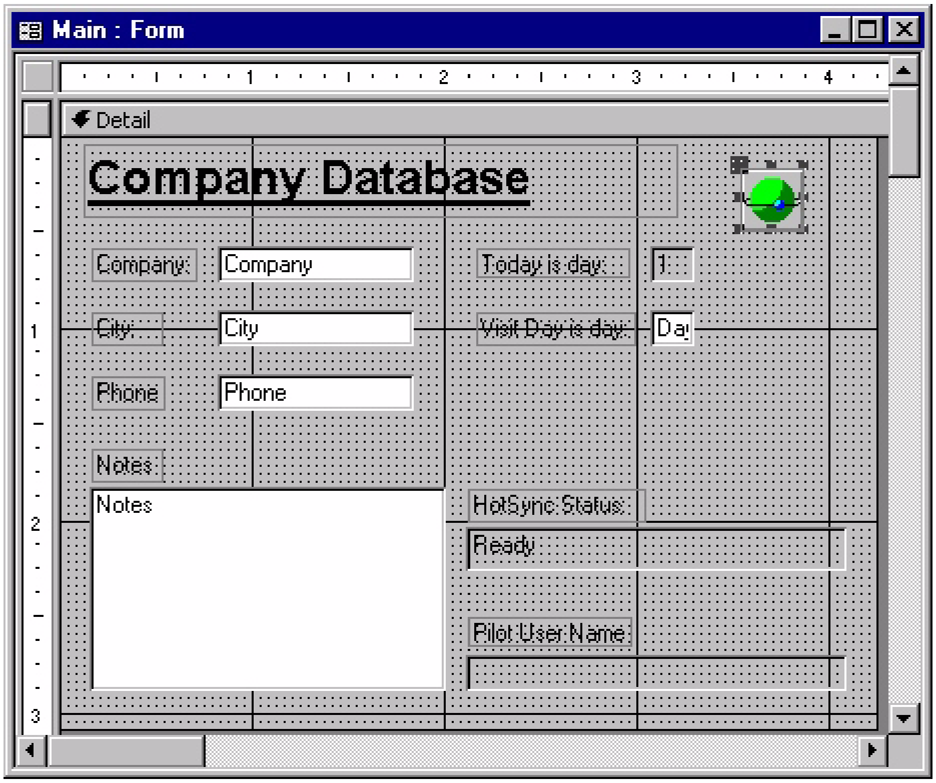
reserved Control
RNTEngine_602 Control
RNTExprVldr Control
RNTMenu
RNTProcessManager Control
RNTSpell Control
RNTTree Control
RotateBvr Class
SagekeySecurity.SecurityCtrl
Satellite Forms 7.0 HotSync Control
SatelliteFormsActiveSync.InkToBitmap

Result

Inserts a new Satellite Forms 7.0 HotSync Control into your document.

Click the **OK** button to add the control to the form. The control is visible during design time, but is invisible when you run your application. Therefore, just place it somewhere out of the way, as shown in the following figure:

Figure 8.3 Access 2000 form with Satellite Forms ActiveX control



Tip By convention, the Satellite Forms ActiveX control in sample code is always named **SatForms**.

A form with an enabled Satellite Forms HotSync ActiveX control receives a control event whenever HotSync is started, when the user presses the HotSync button on the handheld cradle, or when the user initiates a remote HotSync.

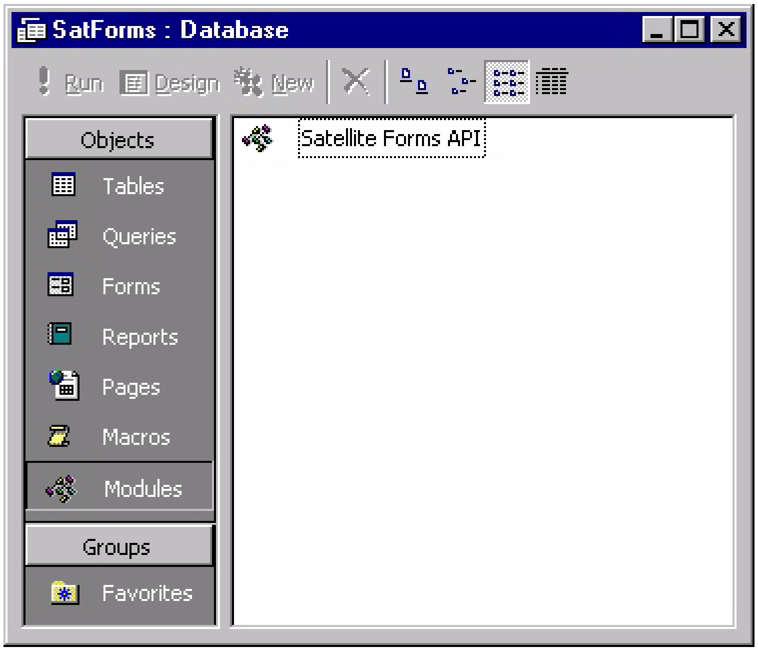
The rest of the HotSync process is the responsibility of the code that you write in your DBMS. This code extracts, merges, or both, information from the desktop DBMS tables and the MobileApp Designer tables and interacts with the Satellite Forms HotSync control to transfer tables and possibly also applications between the desktop and the handheld device.

Satellite Forms includes an Access add-in file: SatForms.mda. This add-in contains useful constants and information that you can use when writing your HotSync handler code. To enable this add-in, open the database you are working with using Access, select **File > Get External Data > Import...** from the Access menu, navigate to the Satellite Forms **Include** directory, click **SatForms.mda**, and click the **Import** button.

If you installed Satellite Forms in the default installation directory, the SatForms.mda file is located in:
C:\Satellite Forms 8\Include\

Click the **Modules** button to see the Satellite Forms API listing, as shown in the following figure:

Figure 8.4 Satellite Forms API for Access



Users of other DBMS products should refer to the **ActiveX.txt** file located in the Satellite Forms Doc directory. The information contained in this file is identical to the information in the SatForms.mda file. Use this file as a starting point when writing the code for your particular DBMS.

If you installed Satellite Forms in the default installation directory, the ActiveX.txt file is located in:
C:\Satellite Forms 8\Doc\

Satellite Forms HotSync ActiveX control events

The Satellite Forms HotSync ActiveX control supports a single event, `HotSyncStatus`. The control fires this event when the user starts a HotSync operation. For the event to fire, the control must be enabled. For instructions, see Satellite Forms HotSync ActiveX control properties on page 207. If the control is not enabled, it ignores the HotSync operation and does not fire an event.

The HotSyncStatus event contains two integer parameters: `StatusCode` and `Param`. `StatusCode` specifies the HotSync event that occurred. `Param` is a generic parameter whose meaning depends on the value of `StatusCode`. For more details, see the description of status codes in Table 8.1.

HotSyncstatus event parameters

The following table lists and describes the HotSync status event parameters.

Table 8.1 HotSync status event parameters

StatusCode	Param	Comments
Status_HotSyncStart (value = 1)	Not used	Indicates that a HotSync operation has started and the Satellite Forms conduit is ready to accept commands. During this event, determine what needs to be done, possibly based on the user name or user ID of the handheld device, and issue any file transfer commands you require. The control provides methods to request file operations. For more information, see the following section, Satellite Forms HotSync ActiveX control for Palm OS.
Status_HotSync-CommandComplete (value = 3)	File transfer result code	Indicates that a file transfer operation has completed. Each file transfer operation you request generate this event when it completes. If the command was successful, `Param` is 1. If the command was unsuccessful, `Param` is 0.
Status_HotSyncEnd (value = 2)	HotSync overall result code	This status code indicates that the entire Satellite Forms HotSync operation completed. The overall result of the HotSync operation is passed in `Param`. If all operations were successful, `Param` is 1. If any or all commands failed, `Param` is 0.

Satellite Forms HotSync ActiveX control methods

The Satellite Forms HotSync ActiveX control provides methods that allow you to copy tables and applications – when you want to install an upgrade – to and from the handheld device and the desktop computer. In most cases, you only need to use these methods while executing code in your `HotSyncStatus` event handler. The methods for file transfer and user management are described in the following pages.

Note These methods only cause a request for a particular operation to be queued with the conduit. On completion of any of these methods, the requested operation has **not** occurred yet. The operation takes place when the conduit regains control, usually when your `HotSyncStatus` event handler is finished. After the transfer operation occurs, a `HotSyncStatus` event fires.

The following table provides an overview of the Satellite Forms HotSync ActiveX control methods:

Table 8.2 Satellite Forms HotSync ActiveX control methods

Method‘	**Description**
CopyAppToPalmPilot	Copies the specified application to the handheld device.
CopyAppToPalmPilotEx	Copies the specified application to the handheld device with a specific Creator ID.
CopyTableToPalmPilot	Copies a specified table to the handheld device.
CopyTableToPalmPilotEx	Copies a specified table to the handheld device with a specific Creator ID.
GetTableFromPalmPilot	Retrieves the specified table from the handheld device.
HsAbandonChanges	Discards all changes made to the HotSync Manager’s user list since the last call to `HsCommitChanges`.*(Deprecated in Satellite Forms 7.2)*
HsAddUser	Adds a new user to the HotSync Manager user list.
HsCommitChanges	Commits all changes made to the HotSync Manager’s user list up to this point.*(Deprecated in Satellite Forms 7.2)*
HsDeleteUser	Deletes a user from the HotSync Manager user list.
HsFindUserByID	Returns information about the specified user.
HsGetFirstUser	Returns information about the first user in the HotSync Manager’s user list.
HsGetNextUser	Returns information about the next user in order in the HotSync Manager’s user list.
HsRenameUser	Changes the user name of a user in the HotSync Manager user list.
InstallPrcFileToPalmPilot	Installs the specified handheld device native file on the specified handheld device.

File Transfer methods

CopyAppToPalmPilot

CopyAppToPalmPilot(*Filename* As String, *CreatorID* As String, *SDDI_DllName* As String, *CreateFlag* As Long, *VersionMajor* As Integer, *VersionMinor* As Integer)

Copies the application specified by *Filename* to the handheld device.

Parameters	*Filename*	The full path and file name of the application to be copied.
	CreatorID	The four-character Creator ID string.
	SDDI_DllName	The full path and file name of the SDDI DLL.
	CreateFlag	The table attribute flag value (see CreateFlag Parameter Values below for more information).
	VersionMajor	Integer specifying the major version of the application.
	VersionMinor	Integer specifying the minor version of the application.

Return Value None

Comments The *Filename* parameter must contain the full path and file name of the application to be copied. This method only copies the application to the handheld device. Be sure to copy all required tables for a new or upgraded application as well. The *CreateFlag* value should be 0 for a standard read/write table, or 2 for a read only table (see Setting the table name and database options on page 105 for more information about table attributes).

This function is not commonly used.

See Also CopyAppToPalmPilotEx, CopyTableToPalmPilot, CopyTableToPalmPilotEx

CopyAppToPalmPilotEx

CopyAppToPalmPilotEx(*Filename* As String, *CreatorID* As Integer)

Copies the application specified by *Filename* to the handheld device with the Creator ID specified by *CreatorID*.

Parameters	*Filename*	The full path and file name of the application to be copied.
	CreatorID	The numeric Creator ID.

Return Value None

Comments The *Filename* parameter must contain the full path and file name of the application to be copied. This method only copies the application to the handheld device. Be sure to copy all required tables for a new or upgraded application as well. This feature is generally only used with the Satellite Forms RDK to group files by creator. For details, see the Satellite Forms RDK Addendum .

This function is not commonly used.

See Also CopyAppToPalmPilot, CopyTableToPalmPilot, CopyTableToPalmPilotEx

CopyTableToPalmPilot

CopyTableToPalmPilot(*Filename* As String, *CreatorID* As String, *SDDI_DllName* As String, *CreateFlag* As Long, *VersionMajor* As Integer, *VersionMinor* As Integer)

Copies the table specified by *Filename* to the handheld device.

Parameters	*Filename*	The full path and file name of the table to be copied.
	CreatorID	The four-character Creator ID string.
	SDDI_DllName	The full path and file name of the SDDI DLL.
	CreateFlag	The table attribute flag value (see Comments below).
	VersionMajor	Integer specifying the major version of the application.
	VersionMinor	Integer specifying the minor version of the application.

Return Value None

Comments The *Filename* parameter must contain the full path and file name of the desktop database table to be copied, eg. C:\MyAppData\Customers.DBF. The *CreateFlag* value should be 0 for a standard read/write table, or 2 for a read only table (see Setting the table name and database options on page 105 for more information about table attributes).

Most desktop sync applications wil use this function.

See Also CopyTableToPalmPilotEx

CopyTableToPalmPilotEx

CopyTableToPalmPilotEx(*Filename* As String, *CreatorID* As Integer)

Copies the table specified by *Filename* to the handheld device with the Creator ID specified by *CreatorID*.

Parameters	*Filename*	The full path and file name of the table to be copied.
	CreatorID	The numeric Creator ID.

Return Value None

Comments The *Filename* parameter must contain the full path and file name of the table to be copied. This feature is generally only used with the Satellite Forms RDK to group files by creator. For details, see the Satellite Forms RDK Addendum .

This function is not commonly used.

See Also CopyTableToPalmPilot

GetTableFromPalmPilot

GetTableFromPalmPilot(*Filename* As String, *CreatorID* As String, *SDDI_DllName* As String, *CreateFlag* As Long, *VersionMajor* As Integer, *VersionMinor* As Integer)

Retrieves the table specified by *Filename* from the handheld device.

Parameters	*Filename*	The full path and file name of the table to be retrieved.
	CreatorID	The four-character Creator ID string.
	SDDI_DllName	The file name of the SDDI DLL, eg. SDDI_PalmDB.DLL.
	CreateFlag	The table attribute flag value (see Comments below).
	VersionMajor	Integer specifying the major version of the application.
	VersionMinor	Integer specifying the minor version of the application.

Return Value None

Comments The *Filename* parameter must contain the full path and file name of the desktop database table to be retrieved, eg. C:\MyAppData\Customers.DBF. The *CreateFlag* value should be 0 for a standard read/write table, or 2 for a read only table (see Setting the table name and database options on page 105 for more information about table attributes).

Most desktop sync applications wil use this function.

See Also CopyTableToPalmPilot, CopyTableToPalmPilotEx

InstallPrcFileToPalmPilot

InstallPrcFileToPalmPilot(*Filename* As String, *HotSyncUserID* As Integer) As Long

Installs the handheld device native file specified by *Filename* on the handheld device specified by *HotSyncUserID*.

Parameters	*Filename*	The full path and file name of the native file to be installed.
	HotSyncUserID	The numeric HotSync User ID.

Return Value TRUE (non-zero) if the method succeeds. FALSE (zero) if the method fails.

Comments The *Filename* parameter must contain the full path and file name of the native file (.PRC or .PDB) to be installed. If a HotSync event is in progress when you call this method, you can set *HotSyncUserID* to zero to install the file on the handheld device currently being synchronized. To install files to a specific handheld device, use the methods described in the following section: User Management methods.

See Also CopyAppToPalmPilot, CopyTableToPalmPilot, CopyTableToPalmPilotEx

Note This method only prepares the file to be installed. The file is actually installed when a HotSync session for the specified handheld is performed to completion. Note that if you call this method from within your `HotSyncStatus` event handler with `HotSyncUserID` set to zero, the native file is installed at the end of the current HotSync session.

User Management methods

These methods allow you to manipulate the user list the handheld HotSync Manager maintains.

Note Prior to Satellite Forms 7.2, changes were not actually made to the user list until you called the `HsCommitUserChanges` method. To discard changes, you would call the `HsAbandonChanges` method. Now, starting with Satellite Forms 7.2, all user changes are carried out immediately, and the `HsCommitUserChanges` and `HsAbandonChanges` methods are simply ignored.

Caution Be very careful when using these methods. If you apply them incorrectly, they may cause irretrievable loss of handheld data. It is highly recommended that you synchronize the handheld device and then back up the handheld directory on your desktop computer while developing applications that use these methods.

HsAbandonChanges

HsAbandonChanges() As Long

Deprecated. Starting with Satellite Forms 7.2, all user changes are carried out immediately, and the HsCommitUserChanges and HsAbandonChanges methods are simply ignored.

Parameters None

Return Value Always returns FALSE.

Comments Prior to Satellite Forms 7.2, this method discarded changes made using any other `Hs*` methods.

See Also HsCommitChanges

HsAddUser

HsAddUser(*HotSyncUserName* As String, *HotSyncUserID* As Long, *HotSyncSubDir* As String) As Long

Adds a new user to the HotSync Manager user list.

Parameters *HotSyncUserID* The HotSync Manager user ID of the user to be added to the list.

HotSyncUserName The HotSync Manager user name of the user to be added the list.

HotSyncSubDir The HotSync subdirectory associated with the user to be added to the list.

Return Value TRUE (non-zero) if the method successfully added the new user to the user list. FALSE (zero) if the method failed to add the new user to the user list.

Comments Call HsCommitChanges to commit the new user’s information to the user list.

See Also HsDeleteUser, HsCommitChanges

HsCommitChanges

HsCommitChanges() As Long

Deprecated. Starting with Satellite Forms 7.2, all user changes are carried out immediately, and the HsCommitUserChanges and HsAbandonChanges methods are simply ignored.

Parameters None

Return Value Always returns TRUE.

Comments Prior to Satellite Forms 7.2, this method commited changes made using any other `Hs*` methods (until you called this method, no changes were made to the HotSync Manager’s user list).

See Also HsAbandonChanges

HsDeleteUser

HsAddUser(*HotSyncUserID* As Long) As Long

Deletes a user from the HotSync Manager user list.

Parameter *HotSyncUserID* The HotSync Manager user ID of the user to be deleted from the list.

Return Value TRUE (non-zero) if the method successfully deleted the user from the user list. FALSE (zero) if the method failed to delete the user from the user list.

Comments Call HsCommitChanges to commit the user deletion from the user list.

See Also HsAddUser, HsCommitChanges

HsFindUserByID

HsFindUserByID(*HotSyncUserID* As Long, *HotSyncUserName* As String, *HotSyncSubDir* As String, *fInstalled* As Long) As Long

Returns information about the user specified by *HotSyncUserID*.

Parameters	*HotSyncUserID*	In. Contains the HotSync Manager user ID of the user whose information is to be retrieved.
	HotSyncUserName	Out. Contains the HotSync Manager user name of the specified user in the list when the method returns.
	HotSyncSubDir	Out. Contains the HotSync subdirectory associated with the specified user in the list when the method returns.
	fInstalled	Out. True (non-zero) is the specified user is installed on a handheld device. False (zero) if the specified user is not installed on a handheld device.

Return Value TRUE (non-zero) if there is information about the specified user in the user list. FALSE (zero) if the specified user is not in the user list.

Comments Pass in empty parameters except for *HotSyncUserID*. This method fills the remaining parameters with information on return.

See Also HsGetFirstUser, HsGetNextUser

HsGetFirstUser

HsGetFirstUser(*HotSyncUserID* As Long, *HotSyncUserName* As String, *HotSyncSubDir* As String, *fInstalled* As Long) As Long

Returns information about the first user in the HotSync Manager's user list.

Parameters	*HotSyncUserID*	Out. Contains the HotSync Manager user ID of the first user in the list when the method returns.
	HotSyncUserName	Out. Contains the HotSync Manager user name of the first user in the list when the method returns.
	HotSyncSubDir	Out. Contains the HotSync subdirectory associated with the first user in the list when the method returns.
	fInstalled	Out. True (non-zero) is the first user is installed on a handheld device. False (zero) if the first user is not installed on a handheld device.

Return Value TRUE (non-zero) if there is information about at least one user in the user list. FALSE (zero) if there is not at least one user in the user list.

Comments Pass in empty parameters. This method fills the parameters with information on return.

See Also HsGetNextUser, HsFindUserByID

HsGetNextUser

HsGetNextUser(*HotSyncUserID* As Long, *HotSyncUserName* As String, *HotSyncSubDir* As String, *fInstalled* As Long) As Long

Returns information about the next user in order in the HotSync Manager's user list.

Parameters	*HotSyncUserID*	Out. Contains the HotSync Manager user ID of the next user in the list when the method returns.
	HotSyncUserName	Out. Contains the HotSync Manager user name of the next user in the list when the method returns.
	HotSyncSubDir	Out. Contains the HotSync subdirectory associated with the next user in the list when the method returns.
	fInstalled	Out. True (non-zero) is the next user is installed on a handheld device. False (zero) if the next user is not installed on a handheld device.

Return Value TRUE (non-zero) if there is information about another user in the user list. FALSE (zero) if there is not another user in the user list.

Comments Call this method after calling `HsGetFirstUser`. Use this method in a loop to retrieve information on all subsequent users. Pass in empty parameters. This method fills the parameters with information on return.

See Also HsGetFirstUser, HsFindUserByID

HsRenameUser

HsRenameUser(*HotSyncUserID* As Long, *NewUserName* As String) As Long

Changes the user name of a user in the HotSync Manager user list.

Parameters

HotSyncUserID The HotSync Manager user ID of the user whose user name you want to change.

NewUserName The new user name for the specified user.

Return Value TRUE (non-zero) if the method successfully changed the specified user name. FALSE (zero) if the method failed to change the user name.

Comments Call HsCommitChanges to commit the user name change to the user list.

See Also HsCommitChanges

Satellite Forms HotSync ActiveX control properties

The Satellite Forms HotSync ActiveX control has several properties that allow you to activate and deactivate the control and to retrieve information about the HotSync operation in progress.

The following table lists and describes the properties of the Satellite Forms HotSync ActiveX control.

Table 8.3 Satellite Forms HotSync ActiveX control properties

Name	Type	Description
Enabled	Boolean	The Satellite Forms control always starts up disabled and does not fire any events during HotSync. To enable the control, your code must explicitly set this property to `True` (1) in the initialization code of the form that contains the Satellite Forms control.
PilotUserName	String	The user name of the handheld being synchronized. This property is read-only and valid only during a `HotSyncStatus` event. (The same UserName value is available in your handheld application using the GetUserName script method.)
UserID	Integer	The Satellite Forms-generated unique user ID of the handheld being synchronized. This property is read-only and valid only during a `HotSyncStatus` event. (The same UserID value is available in your handheld application using the GetUserID script method.) This is NOT the same as the *HotSyncUserId* used by the Hs* methods in the Satellite Forms HotSync ActiveX control.

Table 8.3 Satellite Forms HotSync ActiveX control properties *(Continued)*

Name	Type	Description

CmdType	Integer	The method that just completed. This property contains one of the following values:
		CmdType_Unknown=0
		CmdType_CopyAppToPda=1
		CmdType_Reserved0=2
		CmdType_CopyTableToPda=3
		CmdType_GetTableFromPda=4
		CmdType_Reserved1=5
		CmdType_ReadPdaInfo=6
		CmdType_CopyAppToPdaEx=7
		CmdType_CopyTableToPdaEx=8
		CmdType_CopyPrcToPda=9
		Other values are undefined. This property is read-only and valid only during a HotSyncStatus event with status Status_HotSyncCommandComplete.

CmdFilename	String	The name of the file affected by the method that just completed. This property is read-only and valid only during a HotSyncStatus event with a status of Status_HotSyncCommandComplete.
CreatorID	Integer	This property contains the handheld Creator ID in integer format of the Satellite Forms application currently being synchronized. This integer representation is useful for passing methods to the ActiveX that require the Creator ID as a parameter. If you want to compare against a known Creator ID, use the CreatorString property, which returns the same value in string format.
CreatorString	String	This property contains the handheld Creator ID in string format of the Satellite Forms application currently being synchronized. The string representation is useful If you want to compare against a known Creator ID as described in the Satellite Forms RDK Addendum. If you are using an ActiveX method that requires the Creator ID as a parameter, use the CreatorID property, which returns the same value in integer format. This property is usually only used when deploying an application with the Satellite Forms RDK.

The CreatorID and CreatorString properties are different representations of the same value: the Palm OS Creator ID of the handheld application currently being synchronized.

As explained in the Deploying your Application chapter, the Satellite Forms SDK engine generates a single set of HotSyncStatus events no matter how many applications are actually installed on a handheld device. These notifications are generated with a Creator ID of SMSF. The CreatorString property contains SMSF and the CreatorID property contains 1397576518 in decimal. In hexadecimal, this number is 53 4d 53 46, which converts to the ASCII characters S M S F.

An important difference between Satellite Forms SDK and RDK is that every Satellite Forms application created with the RDK has its own Creator ID and therefore generates its own set of HotSyncStatus events. When deploying applications with the RDK, the `CreatorString` property becomes important in determining whether an event is from your application or from another third-party application also installed on the same handheld.

When using the Satellite Forms SDK, the `CreatorString` property is usually not used.

HotSync and Satellite Forms HotSync ActiveX control

When you use the Satellite Forms HotSync ActiveX control, the control center of the HotSync operation is the code you will write to handle the `HotSyncStatus` event in your desktop DBMS application. The `HotSyncStatus` event handler mediates the transfer of information between the handheld and your database application.

Although the event handler can vary considerably depending on the details of your particular application, most `HotSyncStatus` event handlers operate in the following general way:

1 Event handler entered with `StatusCode = Status_HotSync Start`

Determine the specific handheld device that is being synchronized by examining the `PilotUserName` or `UserID` properties in the Satellite Forms control. Call the `GetTableFromPalmPilot` method as many times as necessary to retrieve any tables that contain information the user may have entered on the handheld device. Then exit the event handler so the conduit can process your requests.

2 Event handler entered with `StatusCode = Status_HotSync CommandComplete`

The event handler is called one time for each call to `GetTableFromPalmPilot` in step 1. Check the `Param` property each time to make sure that the command completed successfully, then exit the event handler immediately except when notified of the last operation requested in step 1. You can determine the last operation by counting notifications or by checking the filename in the property `CmdFilename`.

When you receive notification that the last `GetTableFromPalmPilot` method you requested has completed, use commands specific to your particular DBMS to integrate the data from the newly retrieved tables into the tables of your DBMS. With Access, you might use an Update Query for this purpose. For more information, refer to your DBMS documentation.

Next, extract any data you want to download to the handheld from your DBMS tables and put the data into the dBase tables of your Satellite Forms application. The commands required to do this will vary considerably depending on your particular DBMS and application. With MS Access, you might use an Append Query for this purpose. For more information, refer to the DBMS documentation.

Once the dBase tables you want to copy to the handheld are ready, call the CopyTableToPalmPilot method for each table you want to copy and exit the event handler so the conduit can process your requests.

3 Event handler entered with StatusCode = Status_HotSync CommandComplete.

The event handler will be called one time for each time you called CopyTableToPalmPilot in step 2. Check Param each time to make sure that the command completed successfully, then exit the event handler immediately.

4 Event handler entered with StatusCode = Status_HotSyncEnd.

This status code tells you all operations are completed and the Satellite Forms HotSync operation is complete. Check Param to make sure that everything went okay, then exit the event handler. You will receive no more notifications during this HotSync.

Sample desktop synchronization applications are provided in the \Satellite Forms 8\Samples folder. A Visual Basic 6 sample application titled SatSync is available in the \Satellite Forms 8\Samples\SatSync\ folder. Microsoft Access 2000 samples sync applications are provided in these folders:

\Satellite Forms 8\Samples\Projects\Customers\Access 2000\Access\Customers2000.mdb

\Satellite Forms 8\Samples\Projects\Deliveries\Access 2000\Access\Deliv2000.mdb

HotSync without ActiveX

If your desktop DBMS does not support or you prefer not to use ActiveX controls, it is still possible to control HotSync by accessing the API of the Satellite Forms DLL. This method, although functionally equivalent, is considerably more complex and requires knowledge of Windows programming concepts.

The basic idea is that all the properties, methods, and events of the Satellite Forms ActiveX control have low-level API equivalents provided by **SFrmUt80.dll**. The following table shows the mappings.

Table 8.4 Satellite Forms ActiveX control API usage

ActiveX Feature	DLL Equivalent	Comments
HotSyncStatus event	A registered message is broadcast to all top-level windows.	Use `SF_GetRegisteredMsg` to get the value of the message that is broadcast and watch for it in a top-level window of your application. `WPARAM` contains `StatusCode` and `LPARAM` will contains `Param`. Call `SF_AddToCommandQueue` to make your file transfer requests. Every time your message procedure returns, the conduit will check if there are any requests queued. If so, it executes them. When there are no more requests, HotSync terminates.
File transfer methods	Call `SF_AddToCommandQueue` to make your file transfer requests.	Pass in `CmdType` and `CmdFilename` as parameters. Usually called in response to receiving the message above.

Properties:
`PilotUserName`
`UserID`
`CmdType`
`CmdFilename`
`SatFormsDir`
	Call:
	`SF_GetUserName`,
	`SF_GetUserID`,
	`SF_GetCmdType`,
	`SF_GetCmdFilename`,
	`SF_GetSatFormsDir`
		Usually called in response to receiving the message above.

The API this DLL exports is declared in **SatForms.mda** and in the C-language include file **DllApi.h**. These files are installed in the **Include** directory of the Satellite

Forms. The import library for this DLL is named **SFrmUt80.lib** and is located in the **Lib** directory.

Satellite Forms Synchronization for Pocket PC

The Satellite Forms synchronization system for Pocket PC is conceptually similar to the Palm OS system, with some important differences. An obvious difference is that the Palm OS system is based on the Palm Hotsync service, while Pocket PC devices utilize the Microsoft ActiveSync service to manage the connection between the handheld and the desktop. An additional difference between the two systems that is not readily apparent is in the way that database tables are converted between desktop and handheld formats.

With the Palm OS system, the processes of file transfer between desktop PC and handheld, and database file conversion between desktop format (DBF/MDB) and handheld format (PDB) are integrated into a single process. With the Pocket PC system, file transfer and database file conversion are separate steps. Both the file transfer and database conversions steps must be carried out for the synchronization system to work, but what takes a single step on the Palm OS system takes two equivalent steps on the Pocket PC system.

There are two options provided with Satellite Forms to handle the file transfer step for Pocket PC: the Satellite Forms ActiveSync ActiveX control, or a non-ActiveX approach that relies on the CeRemote.DLL. Both methods rely on the SFConvertPDB utility to handle the database file conversion tasks.

SFConvertPDB Utility

SFConvertPDB is a commandline utility included with Satellite Forms that runs on the desktop computer, not on the handheld. It enables you to convert a desktop DBF or MDB database to a Satellite Forms Palm DB (PDB) handheld database, as a step in the overall process of sending data from the desktop to the handheld. It also enables you to convert a Palm DB (PDB) database into a desktop DBF or MDB database, as a step in the overall process of retrieving data from the handheld to the desktop.

SFConvertPDB does not require a Palm HotSync or Microsoft ActiveSync session to be active, as it runs entirely on the PC. This provides a handheld-platform-independent PC based mechanism to convert data to & from PDB files, for use with Satellite Forms applications on the Palm OS platform and on the Pocket PC platform when using PDB database tables.

SFConvertPDB is an EXE executable program, rather than an ActiveX control or DLL. To integrate SFConvertPDB into your synchronization system, your sync application must be capable of launching an EXE application. **SFConvertPDB.exe** is located in the \Satellite Forms 8\Redist\PocketPC\ folder.

SFConvertPDB Usage

SFConvertPDB operates via commandline switches, in which you supply the required information about which database file to convert, and how to convert it.

Usage: `SFConvertPDB [commandline switches]`

-[PDBtoPC|PCtoPDB]: Specifies whether you want to convert from a PDB database to a DBF/MDB database [use -PDBtoPC], or from a PC DBF/MDB database to a PDB [use -PCtoPDB]. You may specify only one of these conversion options, not both.

-filename \path\to\PC_database_file.[DBF|MDB]: You must supply the path and filename of the PC DBF or MDB database file which you wish to convert to or from PDB. You do not specify the PDB file even when converting from PDB to DBF/MDB: rather, you always supply the PC database filename ending in .DBF or .MDB. The PDB file must exist in the same folder as the DBF/MDB file as specified by the path.

-creatorid CRID: Where CRID is the four-character unique Creator ID used by your application, as defined in the Project Properties settings screen in MobileApp Designer. You must supply the correct case sensitive characters, which cannot include spaces. When testing with the SatForms SDK runtime engine used by MobileApp Designer, the creatorid is SMSF. This parameter is needed in order to generate the correct PDB table name for conversion, as the creatorid is incorporated into the PDB table name. The default creatorid if none is supplied is SMSF.

-SDDI_DLL DLLfilename: You must specify the correct SatForms SDDI DLL to perform the conversion. At the present time, the only conversion supported is for SatForms Palm DB (PDB) databases, so you should always specify the SDDI_PalmDB.DLL. This parameter is optional, and if you omit it, the default of SDDI_PalmDB.DLL will be used.

-VersionMajor VV: Where VV is the major version number of your application, as defined in the Project Properties settings screen in MobileApp Designer. The allowable values are from 0 - 99. This parameter is needed in order to generate the correct PDB table name for conversion, as the major and minor version numbers are incorporated into the PDB table name. This parameter is optional, and if you omit it, the default value of 0 will be used.

-VersionMinor vv: Where vv is the minor version number of your application, as defined in the Project Properties settings screen in MobileApp Designer. The allowable values are from 0 - 99. This parameter is needed in order to generate the correct PDB table name for conversion, as the major and minor version numbers are incorporated into the PDB table name. This parameter is optional, and if you omit it, the default value of 0 will be used.

-CreateFlag n: Where n specifies the desired table attributes (flags) to set on the PDB table when it is created by SFConvertPDB. The allowable values are any positive integer, corresponding to the combination of desired table flags. Certain PDB table behaviours can be set via this numeric flag value, including Backup, Read-Only, and NoAutoCommit flags. The table flags cause the SatForms runtime engine to treat the table differently on the PDA. For a more detailed reference about table flags, see the explanation below. This parameter is optional, and if you omit it, the default value of 0 will be used.

-quiet: This parameter instructs SFConvertPDB to not display any popup error messages when performing the conversion. The exit code of SFConvertPDB can be queried to determine if the conversion was successful or not, making it suitable for calling from sync applications or batch files. When run interactively, you would not likely use the -quiet switch so that you could see error messages pop up if there are conversion problems.

Table Flag Values for the CreateFlag Parameter

The Table Flag is a numeric value that determines special behaviours of that table when it is in use on the PDA by the Satellite Forms runtime engine. For example, one of the possible table flag values indicates that the table is Read-Only, and the runtime engine therefore prevents any modifications/additions/deletions to the data in the table. For more information about these attributes, see Setting the table name and database options on page 105 which explains how to set these table options in the MobileApp Designer Table Editor.

Each table can have different flags as they are assigned on a per-table basis. The current supported table flags include:

Flag Value	Flag Name	Flag Description
0	none	Regular read/write table, with no special behaviours
1	Backup	The table will be backed up at Hotsync (PalmOS only, ignored on Pocket PC)
2	Read Only	The runtime engine will prohibit table modifications/ additions/deletions and will only allow read access to table data. This option applies to both Palm OS and Pocket PC. On Pocket PC there is a performance advantage to using read only tables when possible.
4	autoname	The desktop table name will automatically match the logical table name (*Link table name to filename* option in MobileApp Designer table editor). This flag does not affect behaviour on the handheld at all, it is used by MobileApp Designer table editor only.
64	noautocommit	The NoAutoCommit option means that the table can be modified like a regular read/write table, but that none of the changes to the table are saved automatically. In order to save changes to a NoAutoCommit table, your application has to save or "commit" those changes in script by calling the Tables("tablename").CommitData method. (Pocket PC platform only)

Some table flag values can be combined, while others must be exclusive. For example, the Backup, Read Only, and Autoname flags can all be present on a given table, and you would specify all of those flags by adding their flag values together ([Backup] 1 + [Read Only] 2 + [Autoname] 4 = 7). The Read Only and NoAutoCommit flags must be exclusive to each other: do not combine them together.

SFConvertPDB Sample Usage

Here are some SFConvertPDB usage samples for common conversions scenarios:

- Converting the database file C:\MyApp\Data\EMyAp0102_MYTABLE1.PDB to C:\MyApp\Data\MyTable.DBF

```
SFConvertPDB -PDBtoPC -filename C:\MyApp\Data\MyTable.DBF -
creatorID MyAp -VersionMajor 1 -VersionMinor 2
```

Result: the C:\MyApp\Data\MyTable.DBF is created containing the records from EMyAp0102_MYTABLE1.PDB.

- Converting the database file D:\Server\Data\tNames.MDB to D:\Server\Data\EMyAp0200_TNAMES.PDB with a -CreateFlag value of 7 to indicate Backup + Read-Only + Autoname table flags:

```
SFConvertPDB -PCtoPDB -filename D:\Server\Data\tNames.MDB -
creatorID MyAp -VersionMajor 2 -VersionMinor 0 -CreateFlag 7
```

Result: the D:\Server\Data\EMyAp0200_TNAMES.PDB is created containing the records from tNames.MDB and has the Backup + Read-Only + Autoname table flags set.

Satellite Forms ActiveSync ActiveX control for Pocket PC

The Satellite Forms ActiveSync ActiveX control allows you to interact with the ActiveSync process of the Pocket PC handheld, making it possible for you to copy Satellite Forms tables and applications between desktop computers and handheld devices.

Note The Satellite Forms ActiveSync ActiveX control was originally designed for Pocket PC applications that use the Pocket PC DB (CDB) device database format, but can also be used to sync Palm DB (PDB) databases. Starting with Satellite Forms 8, we support the use of Palm DB (PDB) device database files only for Pocket PC applications (and PalmOS applications), as CDB databases are now obsolete. In addition, you may consider using an alternate non-ActiveX method to synchronize the Pocket PC data with the desktop PC, such as using the SatSyncPPC utility, the CeRemote.DLL, and the SFConvertPDB utility as documented in the Satellite Forms KnowledgeBase. These alternate methods could be used if desired, or you can use the ActiveSync ActiveX control. In both cases (ActiveX or non-ActiveX methods), an integral part of the sync process is the SFConvertPDB database conversion utility described above.

This ActiveSync ActiveX synchronization approach is demonstrated in the sample application **SatSyncPPC**, located in \Satellite Forms 8\Samples\SatSyncPPC.

A typical data synchronization flow using the SF ActiveSync OCX to retrieve a PDB database table from the PocketPC, then send an updated PDB table back to the PPC, would go something like this:

- retrieve the PDB table from the PPC using FileGetFromPPC
- convert the PDB table to MDB/DBF using SFConvertPDB
- manipulate that data in your PC database, creating an updated DBF/MDB
- convert the updated DBF/MDB table to PDB using SFConvertPDB
- download that updated PDB to the PPC using FileSendToPPC

The Satellite Forms ActiveSync ActiveX control is installed and registered on your PC automatically during installation. To use the ActiveX control with your DBMS, you need to place the control on a form in a database application. The following example uses Access 2000.

To place the Satellite Forms ActiveSync ActiveX control onto an Access 2000 form, open the desired database, click the **Objects > Forms** button on the left side of the database window, click the desired form in the list, then click the **Design** button on the database window toolbar. Then select **Insert > ActiveX Control...** from the Access 2000 menu and scroll to and click **SatelliteFormsActiveSync.SFAxPPC61**. Click the OK button to add the control to the form. The control is visible during design time, but is invisible when you run your application. Therefore, just place it somewhere out of the way.

Tip By convention, the Satellite Forms ActiveSync ActiveX control in sample code is always named **SFAxPPC**.

A form with an enabled Satellite Forms ActiveSync ActiveX control receives a control event whenever the Pocket PC device is connected to the PC by placing it in the cradle.

The rest of the sync process is the responsibility of the code that you write in your DBMS. This code extracts, merges, or both, information from the desktop DBMS tables and the MobileApp Designer tables and interacts with the Satellite Forms ActiveSync control to transfer tables and possibly also applications between the desktop and the handheld device.

Satellite Forms ActiveSync ActiveX control events

The Satellite Forms ActiveSync ActiveX control supports two events, `Connected` and `Disconnected`. The control fires the `Connected` event when the Pocket PC device is connected to the PC. Use this event to start the sync process in your sync application. The control fires the `Disconnected` event when the Pocket PC device is disconnected from the PC.

All the functions are active **after** this event.

Satellite Forms ActiveSync ActiveX control methods

The Satellite Forms ActiveSync ActiveX control provides methods that allow you to copy files to and from the handheld device and the desktop computer. In most cases, you will use these methods while executing code in your `Connected` event handler. The methods for file transfer and other functions are described in the following pages.

The following table provides an overview of the Satellite Forms ActiveSync ActiveX control methods:

Table 8.5 Satellite Forms ActiveSync ActiveX control methods

Method	Description
DatabaseToPPC	Sends a desktop database (MDB|DBF) to the Pocket PC and converts it to Microsoft Pocket PC DB handheld database (CDB) format. **Obsolete: CDB databases are no longer supported, use PDB databases and FileSendToPPC function instead.**
DatabaseFromPPC	Retrieves a Microsoft Pocket PC DB device database (CDB) from the Pocket PC and converts it to desktop database (MDB|DBF) format on the PC. **Obsolete: CDB databases are no longer supported, use PDB databases and FileSendToPPC function instead.**
FileSendToPPC	Sends a file from the PC to the Pocket PC without conversion. **Use this function to transfer PDB databases and other files to the Pocket PC.**
FileGetFromPPC	Copies a file from the Pocket PC to the desktop PC without conversion. **Use this function to retrieve PDB databases and other files from the Pocket PC.**
FileExists	Tests whether a file exists on the Pocket PC.

Table 8.5 Satellite Forms ActiveSync ActiveX control methods

Method	Description
FileDelete	Deletes a file from the Pocket PC device.
CreateFolder	Creates a folder on the Pocket PC device.
RemoveFolder	Removes (deletes) a folder from the Pocket PC device.
StartApp	Starts an application on the Pocket PC remotely from the desktop PC.
CheckPassword	Checks the user password on the Pocket PC device to test if it matches the passed string.
ActiveSync_GetActiveSyncFolder	Returns the path where Microsoft ActiveSync is installed to on the desktop PC.

File Transfer methods

DatabaseToPPC

Comments **Obsolete: CDB databases are no longer supported; use PDB databases and FileSendToPPC function instead.**

See Also DatabaseFromPPC, FileSendToPPC

DatabaseFromPPC

Comments **Obsolete: CDB databases are no longer supported; use PDB databases and FileGetFromPPC function instead.**

See Also DatabaseToPPC, FileGetFromPPC

FileSendToPPC

Function FileSendToPPC(*PCFile* As String, *PDAFile* As String) As Long

Sends a file from the PC to the Pocket PC without conversion.

Parameters *PCFile* The full path and file name of the file on the desktop PC to be sent to the Pocket PC handheld.

PDAFile The full path and name of the file as it will be created on the Pocket PC.

Return Value Error code as a Long Integer. An error code of 0 indicates the command completed successfully.

Comments **Use this function to send PDB databases from the desktop PC to the Pocket PC handheld.** This function sends files of any type from the PC to the Pocket PC without performing any conversion.

On Pocket PC 2003 and later versions, a bug in the PocketPC operating system prevents you from overwriting an existing file on the PDA, even if that file is closed and not in use. Therefore, you must delete the file from the PDA first before sending the new file. For example:

```
If SFAxPPC.FileExists(PDAfile) = 0 Then SFAxPPC.FileDelete
(PDAfile)
```

See Also FileGetFromPPC, FileExists, FileDelete

FileGetFromPPC

Function FileGetFromPPC(*PCFile* As String, *PDAFile* As String) As Long

Copies a file from the Pocket PC to the desktop PC without conversion.

Parameters *PCFile* The full path and file name of the file on the desktop PC which will be copied from the Pocket PC handheld.

PDAFile The full path and name of the file on the Pocket PC that will be copied to the desktop PC.

Return Value Error code as a Long Integer. An error code of 0 indicates the command completed successfully.

Comments **Use this function to retrieve PDB databases from the Pocket PC handheld to the desktop PC.** This function gets files of any type from the Pocket PC to the desktop PC without performing any conversion.

See Also FileSendToPPC

FileExists

Function FileExists(*PDAFile* As String) As Long

Tests whether a file exists on the Pocket PC.

Parameters *PDAFile* The full path and name of the file on the Pocket PC to be tested for existence.

Return Value Error code as a Long Integer. An error code of 0 indicates the file exists, otherwise the file does not exist or there was an error.

Comments Note that this function returns a 0 if the file exists, which may be opposite to what you consider the logical result.

See Also FileDelete

FileDelete

Function FileDelete(*PDAFile* As String) As Long

Deletes a file from the Pocket PC device.

Parameters *PDAFile* The full path and name of the file on the Pocket PC to be deleted.

Return Value Error code as a Long Integer. An error code of 0 indicates the file was successfully deleted.

Comments A file that is currently open (in use) on the Pocket PC device will not be deleted

See Also FileExists, DatabaseToPPC, FileSendToPPC

CreateFolder

Function CreateFolder(*Foldername* As String) As Long

Creates a folder on the Pocket PC device.

Parameters *Foldername* The full path of the new folder to be created on the Pocket PC device.

Return Value Error code as a Long Integer. An error code of 0 indicates the folder was successfully created.

Comments User folders are often created as subfolders of \My Documents\ on the Pocket PC device. The \Program Files\ folder is a good location to create your application folder in, for example \Program Files\My App.

See Also RemoveFolder

RemoveFolder

Function RemoveFolder(*Foldername* As String) As Long

Removes (deletes) a folder from the Pocket PC device.

Parameters *Foldername* The full path of the folder to be removed on the Pocket PC device.

Return Value Error code as a Long Integer. An error code of 0 indicates the folder was successfully created.

Comments User folders are often created as subfolders of \My Documents\ on the Pocket PC device. The \Program Files\ folder is a good location to create your application folder in, for example \Program Files\My App.

See Also CreateFolder, FileDelete

Other ActiveSync Control methods

StartApp

Function StartApp(*PDAAppName* As String, [*Commandline* As String]) As Long

Starts an application on the Pocket PC remotely from the desktop PC.

Parameters *PDAAppName* The full path and file name of the Pocket PC device application to be launched.

Commandline The optional commandline to pass to the device application as it is launched.

Return Value Error code as a Long Integer. An error code of 0 indicates the command completed successfully.

Comments The application will be launched as the foreground application on the device, just as if it had been launched manually by the user.

Tip: This function can be used to remotely install a CAB file on the device, by launching the `wceload.exe` utility and passing the pathname of the CAB file on the device, as illustrated in this example from the KnowledgeBase article 10015:

```
SyncAx1.StartApp("wceload.exe", "\MyAppInstall\SFRPPC61.cab")
```

See Also FileSendToPPC

CheckPassword

Function CheckPassword(*Password* As String) as Long

Checks the user password on the Pocket PC device to test if it matches the passed string.

Parameters *Password* The password string to test.

Return Value Error code as a Long Integer. An error code of 0 indicates the password matched.

Comments Note that this function returns a 0 if the password matches, which may be opposite to what you consider the logical result.

ActiveSync_GetActiveSyncFolder

Function ActiveSync_GetActiveSyncFolder() As String

Returns the path where Microsoft ActiveSync is installed to on the desktop PC.

Parameters *None*

Return Value A string containing the path where Microsoft ActiveSync is installed to on the desktop PC.

Comments Note that this returns the path on the desktop PC, not the handheld.

Satellite Forms ActiveSync ActiveX control properties

The Satellite Forms ActiveSync ActiveX control has several properties that allow you to get and in some cases set information relating to the current connected Pocket PC device. In most cases, you will use these methods while executing code in your `Connected` event handler. The following table lists and describes the properties of the Satellite Forms ActiveSync ActiveX control.

Table 8.6 Satellite Forms ActiveSync ActiveX control properties

Name	Type	Description
State	Boolean	The State property is read-only and returns 0 (false) if the Pocket PC is disconnected, or 1 (true) if the device is currently connected to the desktop PC.
DeviceName	String	The device name of the handheld currently connected. This property is read-only and valid only while the device is connected (State property = 1).
LastError	Long Integer	Returns a SatForms ActiveSync ActiveX error number if a function is not executed successfully.
LastErrorText	String	Returns a SatForms ActiveSync ActiveX error message string if a function is not executed successfully.
LastAPIError	Long Integer	Returns an Microsoft ActiveSync API error number if a function is not executed successfully.
ActiveSync_Present	Boolean	This read-only property returns True if ActiveSync is installed on the desktop PC, False if not installed.
ActiveSync_Path	String	Normally must not be changed, however if the ADOFILTR.DLL is not in the folder of ActiveSync then you must set this value to the folder where ADOFILTR.DLL is located on the desktop PC.
ActiveSync_GuestOnly	Boolean	Returns or sets if Microsoft ActiveSync will accept the connected devices as Guests. When working as a guest, you can browse the files on your mobile device and copy or move information, but you cannot synchronize information. Once you disconnect your device from the desktop computer, settings selected in ActiveSync for the guest device are deleted.
ActiveSync_StartOnConnect	String	Returns or sets the pathname of a desktop PC program that ActiveSync starts after a connection is established.
ActiveSync_StartOnDisConnect	String	Returns or sets the pathname of a desktop PC program that ActiveSync starts after a connection is finished.

ActiveSync without ActiveX

If your desktop DBMS does not support or you prefer not to use ActiveX controls, it is possible to control ActiveSync to send and retrieve databases between the desktop and the Pocket PC handheld by accessing the API of the Satellite Forms CeRemote.dll.

This method, although functionally equivalent, does not support all of the features of the ActiveSync ActiveX control.

The basic idea is that all the methods of the Satellite Forms ActiveX control have API equivalents provided by CeRemote.dll, while the properties and events of the ActiveX control are not duplicated in the DLL.

The API this CeRemote.dll exports is declared in the C-language include file **CeRemoteApi.h**, installed in the **Include** directory of Satellite Forms. The import library for this DLL is named **CeRemote.lib** and is located in the **Lib** directory.

This synchronization approach is demonstrated in the sample application **SatSyncPPC**, located in \Satellite Forms 8\Samples\SatSyncPPC.

A typical data synchronization flow using the CeRemote.dll to retrieve a PDB database table from the PocketPC, then send an updated PDB table back to the PPC, would go something like this:

- retrieve the PDB table from the PPC using GetFile
- convert the PDB table to MDB/DBF using SFConvertPDB
- manipulate that data in your PC database, creating an updated DBF/MDB
- convert the updated DBF/MDB table to PDB using SFConvertPDB
- download that updated PDB to the PPC using SendFile

Satellite Forms CeRemote.dll methods

The Satellite Forms CeRemote.dll provides methods that allow you to copy files to and from the handheld device and the desktop computer, as well as additional commands to create folders, check if files exist, and so on. The methods included in the CeRemote.dll are described in the following pages.

The following table provides an overview of the Satellite Forms CeRemote.dll methods:

Table 8.7 Satellite Forms CeRemote.dll methods

Method	Description
SendFile	Sends a file from the PC to the Pocket PC without conversion.
GetFile	Retrieves a file from the Pocket PC to the desktop PC without conversion.
SendTable	**(Obsolete - do NOT use)**.
GetTable	**(Obsolete - do NOT use)**.
RemoteInit	Initializes the connection between the Pocket PC and the desktop PC in order to prepare for file transfer or other methods.
RemoteDeInit	De-initializes the connection between the Pocket PC and PC.
RemoveFile	Deletes a file from the Pocket PC.
CeFileExists	Checks whether a file exists on the Pocket PC.
CeDirectoryExists	Checks whether a directory exists on the Pocket PC.

Table 8.7 Satellite Forms CeRemote.dll methods

Method	Description
CeMakeDirectory	Creates a directory on the Pocket PC.

File Transfer methods

SendFile

Function SendFile(*PCFile* As String, *PDAPath* As String) As Integer

Sends a file from the PC to the Pocket PC without conversion.

Parameters *PCFile* The full path and file name of the file on the desktop PC to be sent to the Pocket PC handheld.

PDAPath The full path to copy the file to on the Pocket PC.

Return Value Result code as Integer. A result code of 0 indicates the command completed successfully.

Comments **Use this function to send PDB databases from the desktop PC to the Pocket PC handheld.** This function sends files of any type from the PC to the Pocket PC without performing any conversion.

Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCSendFile.

Use the SFConvertPDB Utility to convert handheld <--> desktop databases.

See Also GetFile

GetFile

Function GetFile(*PDAFile* As String, *PCPath* As String) As Integer

Retrieves a file from the Pocket PC to the desktop PC without conversion.

Parameters *PDAFile* The full path and file name of the file on the Pocket PC to be copied to the desktop PC.

PCPath The full path to copy the file to on the desktop PC.

Return Value Result code as Integer. A result code of 0 indicates the command completed successfully.

Comments **Use this function to get PDB databases from the Pocket PC to the desktop PC.** This function gets files of any type from the Pocket PC to the desktop PC without performing any conversion.

Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCGetFile.

Use the SFConvertPDB Utility to convert handheld <--> desktop databases.

See Also SendFile

SendTable

Comments **Obsolete: Do NOT use; use PDB databases and SendFile function instead.**

See Also SendFile

GetTable

Comments **Obsolete: Do NOT use; use PDB databases and GetFile function instead.**

See Also GetFile

Other CeRemote.dll methods

RemoteInit

Function RemoteInit(*iWait* As Integer) As Boolean

Initializes the connection between the Pocket PC and the desktop PC in order to prepare for file transfer or other methods.

Parameters *iWait* The time in milliseconds to wait for the connection to be established.

Return Value Result code as Boolean. A result code of True indicates the command completed successfully, False indicates the connection could not be established (for example, the Pocket PC was not attached to the PC).

Comments **Use this function first before any file transfer or other methods.** You must establish the remote connection using this function before any of the other methods can be used. Use RemoteDeInit to close the connection when finished transferring files.

Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCInitConnection.

See Also RemoteDeInit

RemoteDeInit

Sub RemoteDeInit

De-initializes the connection between the Pocket PC and the desktop PC once you are done using the file transfer or other methods.

Parameters *None*

Return Value *None*

Comments **Use this method last when you have completed all of the file transfer and other methods.** You must de-establish the remote connection using this method when finished transferring files.

Note: In the SatSyncPPC sample application this method is declared with the name SF_PPCDeInitConnection.

See Also RemoteInit

RemoveFile

Function RemoveFile(*PDAFile* As String) As Integer

Deletes a file from the Pocket PC.

Parameters *PDAFile* The full path and file name of the file on the Pocket PC handheld to be deleted.

Return Value Result code as Integer. A result code of 0 indicates the command completed successfully.

Comments This function deletes files of any type from the Pocket PC.

Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCDeleteFile.

See Also CeFileExists, CeDirectoryExists, CeMakeDirectory

CeFileExists

Function CeFileExists(*PDAFile* As String) As Boolean

Checks whether a file exists on the Pocket PC.

Parameters *PDAFile* The full path and file name of the file on the Pocket PC handheld to be checked.

Return Value Result code as Boolean. A result code of True indicates the file exists, False indicates the file was not found.

Comments Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCFileExists.

See Also RemoveFile, CeDirectoryExists, CeMakeDirectory

CeDirectoryExists

Function CeDirectoryExists(*PDAPath* As String) As Boolean

Checks whether a directory exists on the Pocket PC.

Parameters *PDAPath* The full path of the directory on the Pocket PC handheld to be checked.

Return Value Result code as Boolean. A result code of True indicates the folder exists, False indicates the folder was not found.

Comments Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCDirExists.

See Also CeFileExists, RemoveFile, CeMakeDirectory

CeMakeDirectory

Function CeMakeDirectory(*PDAPath* As String) As Boolean

Creates a directory on the Pocket PC.

Parameters *PDAPath* The full path of the directory on the Pocket PC handheld to be created.

Return Value Result code as Boolean. A result code of True indicates the folder was created, False indicates the folder was not created (for example it may already exist).

Comments Note: In the SatSyncPPC sample application this function is declared with the name SF_PPCMakeDirectory.

See Also CeFileExists, CeDirectoryExists, RemoveFile

Satellite Forms CeRemote.dll result values

Most of the Satellite Forms CeRemote.dll methods are functions that return a numeric value indicating the result.

The following table provides a list of the Satellite Forms CeRemote.dll result values:

Table 8.8 Satellite Forms CeRemote.dll function result values

Result	Description
0	No error, the function completed successfully.
1	General unspecified error

Table 8.8 Satellite Forms CeRemote.dll function result values

Result	Description
2	Connection error
3	Create File error (file may already be open on PDA)
4	Error deleting file
5	File does not exist
6	Could not find the Filter reg key
7	Directory does not exist
8	Unable to load DLL
9	Unable to load function
10	Open file error (file may already be open on PC)
11	Read error
12	Unable to send
13	Write error

The next step

Phase 2 of creating your Satellite Forms application is now complete. The next phase is covered in Deploying your Application, on page 229 .

Chapter 9
Using Satellite Forms on Handheld Devices

This chapter explains how to use the Satellite Forms Engine to run your applications on handheld devices.

Starting the Satellite Forms engine

You can download many Satellite Forms applications to the same handheld device. When you start the Satellite Forms SDK Runtime Engine on the handheld device, you see a list of all the available Satellite Forms applications. With the Satellite Forms Redistribution Kit (RDK) you can create an icon that launches a Satellite Forms application without having to tap the Satellite Forms icon first.

Palm OS

Procedure Use the Satellite Forms Engine to launch a Palm application on your handheld device:

1 Turn on your handheld device.

2 Tap the **Applications** icon. Icons for all the installed handheld applications appear.

3 Tap the **Sat. Forms** icon. The list of installed Satellite Forms applications appears.

Pocket PC

Procedure Use the Satellite Forms engine to launch a Satellite Forms Pocket PC application

1 Tap **Start**, tap **Programs**, and then tap **Satellite Forms**.

A list of the installed Satellite Forms applications appears.

2 Tap the name of the Satellite Forms application you want to run.

Procedure Use the Satellite Forms Engine to launch a Pocket PC application on your handheld device:

Note The Satellite Forms runtime engine, SatForms80.exe, is installed in the Windows directory on the handheld device. The Satellite Forms icon in the Programs group is a shortcut to the engine.

1 Tap **Start**, tap **Programs**, and then tap **File Explorer**.

2 Browse to the folder that contains the Satellite Forms application (a PDA file) or the folder that contains the Satellite Forms application launcher icon (a PRC file).

3 Tap **<*appname*>.PDA** or **<*appname*>.EXE** to launch the application.

Procedure Create a shortcut to an application on the Start menu

1 Locate **<*appname*>.PDA** or **<*appname*>.EXE** for the application for which you want to create a shortcut.

2 Tap and hold the icon for the file to display the context menu and then tap **Copy**.

3 Tap **Start**, tap **Programs**, and then tap **File Explorer**.

4 Using File Explorer, browse to \My Device\Windows\Start Menu

5 Tap the **Edit** menu on the menu bar and then tap **Paste Shortcut**.

6 Rename the shortcut, if desired.

Using the Satellite Forms Applications list

Using the Satellite Forms applications list, you can open an application or delete an application. You can also display the **About Satellite Forms** dialog box.

Opening an application

Procedure Open a Satellite Forms application on a Palm handheld device

1 Tap the **Applications** icon to display the handheld device's application picker screen.

2 Tap the **Sat. Forms** icon to display the list of installed Satellite Forms applications.

3 Tap the desired application name listed under **Select Application to Run**.

Palm OS

Procedure Delete a Satellite Forms application

1 Tap the **Sat. Forms** icon. The list of installed Satellite Forms applications appears.

2 Tap the handheld device's **Menu** icon.

3 Tap **Delete App**.

4 Tap the application you want to delete from the Satellite Forms Application list.

5 Tap **OK** to confirm the deletion.

Palm OS

The **About Satellite Forms** dialog box shows which version of the Satellite Forms Engine is installed on the handheld device.

Procedure Display the **About Satellite Forms** dialog box on a Palm OS device:

1 Tap the **Applications** icon to open the handheld device's applications list.

2 Tap the **Sat. Forms** icon.

3 Tap the handheld device's **Menu** icon.

4 Select **Options > About** from the menu to display the About Satellite Forms dialog box.

5 Tap **OK** to close the dialog box.

Palm OS Satellite Forms application menus

When you run a Satellite Forms application, the application's initial form opens. For information about setting the initial form, see Project Properties dialog box on page 83.

Tap the handheld device's **Menu** icon to display the three menus available for use with a running application: **Records**, **Edit**, and **Options**. The **Records** menu list is automatically displayed when you tap the handheld device's **Menu** icon. You can use the options on any of these menus by tapping them with the stylus or writing **Command Stroke** plus the assigned command letter in the Graffiti writing area.

The following tables list and describe the menu options available on every Satellite Forms application menu:

Table 9.1 Satellite Forms Records menu

Command	Command Letter	Action
Goto First	F	Goes to the first record of the linked table.
Goto Prev	R	Goes to the previous record of the linked table. A beep sounds if there are no previous records.
Goto Next	N	Goes to the next record of the linked table. A beep sounds if there are no more records.
Goto Last	L	Goes to the last record of the linked table.
Create	T	Creates a new record in the linked table and displays it. The form's properties must have the **Create Record** permission enabled.
Delete	D	Deletes the current record of the linked table. The form's properties must have the **Delete Record** permission enabled.

Table 9.2 Satellite Forms Edit menu

Menu Item	Command Letter	Action
Undo	U	Reverses the last action (cut/copy/paste).
Cut	X	Cuts the selected item and places it on the clipboard.
Copy	C	Copies the selected item onto the clipboard.
Paste	P	Pastes the clipboard contents at the cursor.
Select All	S	Selects all of a control's contents.
Keyboard	K	Displays the handheld's on-screen keyboard. The keyboard only appears if the current form contains Edit or Paragraph controls. Otherwise, a beep sounds.
Graffiti	G	Opens Graffiti Help, a series of screens that show the complete Graffiti penstroke character set.

Table 9.3 Satellite Forms Options menu commands

Menu Item	Command Letter	Action
Exit	Q	Exits the Satellite Forms application and displays to the Satellite Forms applications list.
About		Displays the **About Satellite Forms** dialog box.

Chapter 10
Deploying your Application

This chapter lists and describes steps required for deploying your Satellite Forms application. It is organized into the following sections:

- Overview of deploying applications for Palm OS and Pocket PC devices
- Step-by-step instructions on deploying an application for Palm OS handheld devices
- Step-by-step instructions on deploying an application for Pocket PC handheld devices
- Guidelines for creating a custom installer for your application

Overview

After you have developed and tested your Satellite Forms application, the next step is to distribute your application to your end users. Satellite Forms includes a Re-Distribution Kit (RDK) that contains software components and utilities to enable you to easily deploy the application you created. This chapter contains details about working with the RDK, including step-by-step instructions on how to use it to deploy your application smoothly and special considerations for the deployment phase.

Deploying your application is the third and final phase of developing your Satellite Forms application. This phase involves the following steps, separated by target platform below:

Procedure Deploying Palm OS applications

1 Create and assign a launcher icon to your application.

2 Change the default creator ID to a unique creator ID.

3 Modify the HotSyncStatus Handler.

4 Update applications created with earlier version of Satellite Forms.

5 Set up the redistribution kit.

6 Install Satellite Forms RDK components on your end user’s desktop computers.

7 Install the Satellite Forms runtime engine on your end user’s handheld devices.

8 Install your application on your end user’s handheld devices.

Procedure Deploying Pocket PC applications

1 Create and assign a launcher icon to your application.

2 Rebuild the application in MobileApp Designer.

3 Modify the ActiveSync event handler application.

4 Set up the redistribution kit.

5 Install Satellite Forms RDK components on your end user’s desktop computers.

6 Install the Satellite Forms runtime engine on your end user’s handheld devices.

7 Install your application on your end user’s handheld devices.

Deploying Palm OS applications

This section lists and describes the steps to deploy a Palm OS application. After you have completed the steps in this section, your application will be distributed to your end users and your development cycle will be complete.

Integrated Runtime

Starting with Satellite Forms Version 8, a new integrated runtime engine feature has been added to improve application deployment. This feature works by combining your application launcher icon with the Satellite Forms runtime engine executable, to create a customized runtime branded specifically for your application. Previously, the launcher icon app was a small “loader” customized with your icon that would load the standard RDK runtime engine to run your application. Now the launcher icon and runtime engine are integrated together into one.

The integrated runtime improves app deployment by having one less file to install as a part of your package. More importantly, it makes your deployed app more robust by reducing the chance that installing another Satellite Forms application might affect your installed application by overwriting the runtime engine with a different version. For the Palm OS platform, the RDK runtime engine and you icon are combined into your application PRC file.

In order to have MobileApp Designer generate an integrated runtime engine for your application, you must specify a proper launcher icon in the Project Properties, as described below..

Create and assign application icons

You can specify large and small icons in color or black and white for your application. The icons for your project will appear as small images that represent your application to end users on their devices. This section includes information on creating and assigning the icons to your application.

For an application with a complete set of icons, create the following:

- Large black and white icon: 31x21 pixels. Required for every application.
- Small black and white icon: 15x9 pixels.
- Large color icon: 31x21 pixels.
- Small color icon: 15x9 pixels.

- Large high density color icon: 62x42 pixels.
- Small high density color icon: 30x18 pixels.

⚠ Caution Your application must have the black and white large (31x21) icon associated with the project before it is compiled and distributed. Omitting this icon will cause problems with your application. Although not recommended, the small icon and color icons can be omitted. A “missing” icon will appear where the images would have been in your application on the device.

Guidelines to follow when creating your icons:

- All files must be in bitmap format (bitmap files have a **.bmp** extension).
- After you have chosen a name for the large black-and-white icon, all other icons for your application must match the same naming convention. For example, if you name the large black and white icon *MyIcon*.bmp, you must name the rest of the icon files are shown below:

  Small black and white icon: *MyIcon*-Small.bmp

  Large color icon: *MyIcon*-Color.bmp

  Large high density color icon: *MyIcon*-Color-HD.bmp

  Small color icon: *MyIcon*-Small-Color.bmp

  Small high density color icon: *MyIcon*-Small-Color-HD.bmp
- All icon files for your application should be placed together in the same directory, ideally in a directory specifically for icons or images associated with your application. Satellite Forms will automatically “search” for missing icon files in the directory where the large black and white icon is located.
- The conversion of a color icon to a Palm icon can cause the icon to look different on the device than in the program you use to create the icon. Avoid this problem by using Palm’s official color palette for Palm OS devices. Satellite Forms also includes two color icon template files you can edit to create your own color icons. Navigate to the **Templates** directory and make a copy of the **Template-Color.bmp, Template-Color-HD.bmp, Template-Small-Color.bmp**, and **Template-Small-Color-HD.bmp** files. Paste them into your project directory and edit them with a image editing tool to create your own icon.
- Most software that will generate a bitmap (.bmp) file will work for creating icons. Many computers running the Microsoft Windows operating system will already have the Microsoft Paint program included. This program will generate bitmaps you can use as icons for your application. Some tips on creating icons with this common program are given below as a convenience; refer to the complete Microsoft documentation for complete details on the program:

Tips for creating icons

  - Open one of the template files noted above in Microsoft Paint (File<Open).
  - Draw the icon within the box. Set the background to Transparent. Areas that are set to transparent will be invisible on the handheld device. The default color indicating transparency in Microsoft Paint is typically bright green.
  - The pencil tool is frequently the best tool for creating an icon; zoom in if needed.

- The easiest way to change to a different color from the official Palm palette (the color blocks on the bottom of the template) is with the Color Picker tool. Select the Color Picker, then click to select the color you want to use in the Palm palette. Remember to avoid using any colors other than those in the Palm palette.
- When saving your icon, there are several options available. Save a black and white icon as a Monochrome Bitmap. Save a color icon as a 24-bit Bitmap.

⚠ Caution Do not resize the template file. Each template file is already the correct size.

Change the default creator ID

Each Palm application must have a unique Creator ID. Making sure that your application has a unique Creator ID prevents potential conflicts with other applications on your users' devices. If your application has the Creator ID "**SMSF**" you must obtain your own Creator ID; this is the default value reserved for Satellite Forms. Unique Creator ID's can be obtained from the ACCESS Systems Americas (formerly PalmSource, Inc.) website free of charge.

Once you have a unique Creator ID, you must associate it with your application and rebuild the application.

Procedure Changing your Creator ID and rebuilding your application

1 Navigate to File<Project<Properties.

The Properties dialog box opens.

2 Type your unique Creator ID in the text box for Creator ID's.

3 Save the project and rebuild to generate a new set of Palm binary files.

Your unique Creator ID is now part of your project.

Modify the HotSyncStatus Handler

Before you deploy your application on Palm OS devices, you must add a short section of code to your HotSyncStatus handler. This will ensure that your application can coexist with any other application on the device that was created with Satellite Forms. A brief explanation of potential conflicts is given below, along with sample code.

Each application installed with the Satellite Forms RDK generates its own set of HotSyncStatus events that are broadcast to all applications running on your server. The Satellite Forms RDK engine also generates an additional set of notifications for all applications installed with the RDK.

When you worked with the SDK engine during the testing step, you dealt exclusively with the notifications generated by the SDK engine. The SDK engine is a simplified tool for testing only; since there were no RDK applications installed on your handheld device, you did not have to verify that the HotSyncStatus notifications were coming from one of your applications. In an actual deployment environment, a precautionary step is needed.

When deploying with the RDK, you must add a short section of code to the top of your HotSyncStatus handler to check if a HotSyncStatus originated from your application.

The code should ensure that your HotSyncStatus event handler checks the CreatorString property of the Satellite Forms ActiveX control.

If the CreatorString property does not match the creator ID you assigned to the icon that launches your application, the event does not belong to your application and should be ignored. Your HotSyncStatus handler should return immediately without doing any further processing. A typical example of the simple code that needs to be added to the top of your HotSyncStatus handler is given below, with MYAP representing your unique Creator ID:

Example 10.1 HotSyncStatus handler deployment code

```
//Check if this event is coming from our application
//(assumes the creator id of the app is MYAP)

     If (SatForms.CreatorString <> "MYAP") Then
//event not from our app - exit immediately
          Exit Sub
     End If
```

Update older applications

If you have released versions of your application that were created with an earlier version of Satellite Forms, you will already have many of the redistribution steps complete, but you need to take some extra steps to update your application. If this is the first time you will be distributing your application, disregard this section and continue to the next.

Following the procedure below will bring your application up-to-date with the current Satellite Forms version.

Procedure Update your application to the current Satellite Forms version

1 Edit the **Install.ini** file`[PRC_Files]` section to specify the correct PDB and PRC files for your application.

2 Modify your **Install.bat** batch file as follows:

```
RDKINST -instprc Install.ini -condpath SFrmCnxx.dll  -crtreg
CRID -name "your application name" -restarths
```

Replace **SFrmCnxx.dll** with the correct DLL name from the following list:

- SF 4.0 and 4.1: **SFrmCn40.dll**
- SF 5.0 and 5.1: **SFrmCn50.dll**
- SF 6.0.x: **SFrmCn60.dll**
- SF 6.1: **SFrmCn61.dll**
- SF 7.x: **SFrmCn70.dll**

3 Modify your HotSync program to use new control and new API call improvements in this version of Satellite Forms as follows:

a Replace the old ActiveX control with the new version included with your current Satellite Forms version.

b Modify the CopyTableToPalmPilot and GetTableFromPalmPilot calls as shown:

```
CopyTableToPalmPilot(Filename As String,CreatorID As String, SDDI_Plugin_Name As
String, CreateFlag As Integer,VersionMajor As Integer,VersionMinor As Integer)
```

```
GetTableFromPalmPilot(Filename As String,CreatorID As String, SDDI_Plugin_Name
As String,CreateFlag As Integer,VersionMajor As Integer,VersionMinor As Integer)
```

Where the following term definitions apply:

- **Filename**: Full path to the intermediate database, either MDB or DBF. The file name must be upper case.

  Example: `C:\MyApp\TABLE1.MDB`

- **CreatorID**: Your application's unique four character string Creator ID.
- **SDDI_Plugin_Name**: This should always beset to **Sddi_PalmDB.dll**
- **CreateFlag**: This should be set to 0 for most tables, but could be set to a different value for read only tables, see Table Flag Values for the CreateFlag Parameter on page 213 for complete details.
- **VersionMajor**: Your application's major version number (i.e. **1**.x).
- **VersionMinor**: Your application's minor version number (i.e. 1.**x**).

Tip If you aren't sure what your CreatorID, VersionMajor and VersionMinor values are, open your application in MobileApp Designer, go to Edit > Project Properties, and view the values in the Project Properties dialog.

When you have finished updating your API calls, they should resemble the following:

```
call CopyTableToPalmPilot("C:\MyApp\ORDER.MDB","CRID",
"Sddi_PalmDB.dll",0,1,0)
```

```
call GetTableFromPalmPilot("C:\MyApp\ORDER.MDB","CRID",
"Sddi_PalmDB.dll,0,1,0)
```

With the following values applied:

- **Table**: Order and located at C:\MyApp folder
- **CreatorID**: CRID
- **CreateFlag**: 0
- **Major version**: 1
- **Minor version:** 0

4 Your update is now complete. Continue through the next sections to verify that your updated application is ready to distribute.

Set up the redistribution kit

In this step, you will create several new directories and fill them with subdirectories and file copied over from the Satellite Forms program files directory. When you are finished, the new directories will contain all the redistributable components for your application, including Palm OS-specific files, HotSync Handler files, and databases. Your end users will have this new directory structure installed on their systems.

Caution Follow the instructions below carefully; the directory structure and naming must match the directions given exactly for your installation to proceed smoothly.

Procedure Preparing the redistribution kit.

1 Create a new, stand-alone directory named **Redistribution Kit**.

2 Create two directories beneath the Redistribution Kit directory with the following names:

- **Palm**
- **HotSync Handler**.

The directory structure has now been created.

Tip Navigating to the Satellite Forms Program Files directory now will make the rest of the steps in this procedure easier, as you copy and paste files between the Satellite Forms directories and your new redistribution directories. All pathnames given below are from the root of the Satellite Forms directory.

The default location of the Satellite Forms program is C:\Satellite Forms 8\.

3 Copy all files and subdirectories from the following location:

\Redist\Common\Runtime Installer\Disk

Paste the contents into the new **Redistribution Kit** directory created in Step 1. If you are using your own custom installer, place copies of your installer files in this directory now.

4 In the Redistribution Kit directory created in Step 1, navigate to the following location:

Redistribution Kit\program files\Satellite Forms Runtime for PCs\Palm

Locate, open, and edit the **GeneralReadMe.txt** file in a text editor. This is the readme file that will be included in your application. Include any information your users will need to know about your application.

Caution Do not change the name of the readme file. The readme must be named **GeneralReadMe.txt** for the installer to function correctly.

Note Often, handheld device applications do not have any other product documentation apart from the readme you provide. Make sure your users will have any information they may need about installation, upgrading, uninstallation, operation, troubleshooting, and contact information by adding it to the redistribution readme. If your users will be upgrading, the readme is also a good place to notify your user about the procedures they should follow before installing your application.

5 Copy all Palm PRC and PDB files from the following location:

\AppPkg\Palm (mmnn - CRID) Where **mm** is the major version number of your application, **nn** is the minor version number, and **CRID** is your unique Creator ID.

Paste the PRC and PDB files into the new **Redistribution Kit\Palm** directory created in Step 2.

Note that starting with Satellite Forms 8, if your application uses any PRC extension files, those files will be included in your AppPkg folder along with the other application PRC and PDB files. The integrated runtime engine PRC will also be placed in the AppPkg folder. *In previous versions the runtime engine and extension PRC files had to be copied seperately.*

6 Navigate to the following location:

\Redist\Palm

Copy the file named **RDKInst.exe** and paste it into **Redistribution Kit\Palm** directory created in Step 2.

7 Using a text editor, create the **Install.ini** file that will install your application. The Install.ini file is the configuration file that will control which files are included during installation. If you need help getting started with your Install.ini file, a sample Install.ini file is included with the Redist Work Order sample in the **\Samples\Redist\Work Order** directory. An example file of a typical Install.ini file is shown below:

Example 10.2 Sample Install.ini file

```
;
; Installer Section
;
;     KEY               DESCRIPTION
;
;     EngWinTitlewindow title
;     EngInstructuser instructions to appear on the installer dialog
;

[Installer]

EngWinTitle=Your Application Name  Installer
EngInstruct=Ready to copy Your Application Name to your Palm device.\n\nPress the
"Install" button to continue or press "Cancel" to abort.
;
; PRC File Section
;
; Note: Create one key per PRC/PDB file to install. All files must be listed with
keys of the form 'PrcFileX', where X is consecutive integers starting with zero.
;
;     KEY               DESCRIPTION
;
;     PrcFileX   name of file to install (X starts at 0)
;

[PRC_Files]

PrcFile0 = Your_Application_Name.PRC
PrcFile1 = ECRID0000#Your_Application_Name.PRC
PrcFile2 = ECRID0000$Your_Application_Name.PDB
PrcFile3 = ECRID0000_TABLE1.PDB
PrcFile4 = ECRID0000_TABLE2.PDB
…
PrcFileN = ECRID0000_TABLEN.PDB
```

```
'If you are using extensions, specify them here.
'eg. PrcFileX = SFE_Strings.prc
```

8 Create the **Install.bat** file that will register your application's Creator ID with HotSync Manager. If you need help getting started with your Install.bat file, a sample Install.bat file is included with the Redist Work Order sample in the **\Samples\Redist\Work Order** directory.

Your Install.bat file should resemble the following example:

Example 10.3 Sample Install.bat file

```
rdkinst install.ini -instprc -crtreg CRID -condpath SFrmCn80.dll -name
"Application Name" -restarths
```

Where the following term definitions apply:

- **install.ini**: Contains install rules.
- **instprc**: Instructs RDKInst to install Palm files using Install.ini rules file.
- **crtreg CRID**: Registers a HotSync custom conduit for Creator ID CRID.
- **condpath SFrmCn80.dll**: Specifies the conduit DLL to use for CRID (do NOT specify a path, just the DLL filename).
- **name "Your application name"**: Specifies the name for this new custom conduit.
- **restarths**: Instruct RDKInst to restart HotSync.

9 Copy your backend database, all intermediate database files, and your HotSync Handler program into the **Redistribution Kit\Hotsync Handler** folder created in Step 2.

Note Intermediate databases are created by MobileApp Designer, and can be found at the same level as your project source files.

10 Your application, consisting of the new Redistribution Kit directory created in Step 1 and all files and subdirectories you added within the new directory, is now complete and ready to distribute to your end users. Prepare your application for distribution by completing one of the following steps:

- **Network Installation**: Copy the Redistribution Kit directory and all contents onto a network hard drive.
- **CD Installation**:Copy the Redistribution Kit directory and all contents onto a CD-ROM.

Distributing the application

This section provides step-by-step instructions on distributing your application to your users.

Caution A well-planned deployment should include testing the distribution steps below yourself, with a clean computer and device, before beginning distribution to a wider user audience. At a minimum, you should confirm that your application installs successfully on both the computer and the device, and that your HotSync Handler functions correctly.

Verify end user system requirements

Your Satellite Forms application requires a variety of different minimum system requirements to function smoothly. Verify that all system requirements given below have been met by your user’s equipment before proceeding with installation. If you cannot verify the system requirements for your end users, be sure that you transfer the system requirements listed below to the general redistribution readme (see the readme step in Preparing the redistribution kit.)

Desktop computer requirements

Satellite Forms has been tested with the following operating systems:

- Microsoft Windows Vista Business edition
- Microsoft Windows XP Professional edition
- Microsoft Windows 2000 Professional edition
- Microsoft Windows NT 4.0 Workstation with SP 6
- Windows ME
- Windows 98 Second Edition

Handheld device requirements

The following list describes the recommended hardware and software minimum requirements for the handheld devices on which the Satellite Forms runtime engine is installed.

- Palm OS 3.5 or later
- 1 MB available main memory
- HotSync 3.01 or higher

MSI executable files

End users running Windows 9x or Windows NT need specific MSI executable files before the Satellite Forms components are installed on their computer (newer versions of Windows have these Windows Installer files already present). Run the executable files on each computer running these two operating systems before proceeding with installation. The appropriate executable files are included in the RDK at the following locations:

Windows 9x: \MsiInstall 9x

Windows NT: \MsiInstallNT

MDAC requirements

Microsoft Data Access Components (MDAC) is software that Satellite Forms uses to interface with transfer tables. End users running any of the following software will need to upgrade to MDAC version 2.6 SP1, or later:

- Windows 98
- Windows NT
- Windows 2000

- Access 2000

End users running Windows XP or Access 2002 will already have a compatible MDAC version.

Upgrade each computer to the appropriate version before proceeding with installation.

Note If you install or update the MDAC version, you may also need to update the Microsoft Jet 4.0 engine. Refer to the Microsoft product documentation and website for additional details.

Upgrading previous releases of your application

If your users will be upgrading from earlier versions of your application, the following steps must be completed before installing the new version:

Procedure Preparing user computers for a new application version

1 Unregister the older application's Creator ID with HotSync manager.

Satellite Forms 3.x: use RDKInstS.exe

Satellite Forms 4.x or higher: use RDKInst.exe

To unregister the older version of your application, create a batch file with the following command lines:

Satellite Forms 3.x: `RDKInstS -remreg CRID`

Satellite Forms 4.x or higher: `RDKInst -remreg CRID`

Where CRID is the older application's Creator ID.

If the earlier version of your application was installed with a custom installer, undo the registries created by your custom installer.

⚠ Caution Failure to unregister the older application's Creator ID can cause registry conflict between old and new versions of your application. As a result, your HotSync program will not receive any events that come from your application.

2 Uninstall any Satellite Forms redistribution components that were installed on the user's computer with the earlier version of the application. On a Windows user computer, uninstallation can be completed by navigating to Start > Settings > Control Panel > Add or Remove Programs and selecting the appropriate applications from the list of installed programs. If you are using a custom installer, use your installer to complete the uninstall.

If you cannot verify that previous releases have been uninstalled, be sure to transfer the information above to the general redistribution readme (see the readme step in Preparing the redistribution kit.).

Installing redistributable components

You end users will need to transfer information between your application on their handheld device and their computer. The install process for your application includes installing several components on your end user's desktop computers. This step must be completed before installing the application itself on your user's handheld devices.

These redistributable components are sometimes referred to as the **Satellite Forms Runtime for PCs**. In previous releases this package was named the Satellite Forms Runtime for *Palm*, but with the adoption of the Palm DB (PDB) handheld database format for the Pocket PC platform the runtime package has been renamed.

The Satellite Forms Runtime for PCs can be installed on the end user's computer *interactively* (in which the end user sees the SatForms installer prompts and must click several buttons to either complete or cancel the installation) or *silently* (in which the components are installed and registered without presenting any prompts or interface to the end user). Select the interactive or silent install method according to your needs, described below.

Installing redistributable components *interactively*

Complete the following procedure on each user's computer.

Procedure Interactively install redistributable components on user computers

1 Complete the option that applies to your distribution format:

- **Network install**: Navigate to the Redistribution Kit directory and double-click **Setup.exe**.
- **CD-ROM install**: Double-click **Setup.exe** at the root of the CD.
- **Custom installer**: Run your installer.

The installer program starts.

2 Follow the directions of the installer to complete installation of the Satellite Forms redistributable components. Repeat steps 1 and 2 on each user computer.

Your end user's computers now have the necessary components to communicate with your application.

Installing redistributable components *silently*

Complete the following procedure on each user's computer.

Procedure Silently install redistributable components on user computers

1 Use the Windows Start > Run function to run the setup program with some commandline switches that specify silent install mode. Click on Start > Run and then click the Browse button which opens a standard file selection dialog. Navigate to the Redistribution Kit directory on your installation media (eg. CD or network folder), click on **Setup.exe**, then click on Open. That will return you to the Run dialog with the full path to the Setup.exe file shown in quotes.

2 Move the cursor to the end of that line without overwriting the path to Setup.exe, and add the commandline switches **`/S /v/qn`** to specify the silent mode, leaving a single space after the end quote, before the switches. You must enter these commandline switches *exactly as shown*: the quoting and spacing matters.

An example commandline shown in the Run dialog, assuming a CD install media using drive letter X: would be:

```
"X:\Redistribution Kit\Setup.exe" /S /v/qn
```

3 Click on OK. The Run dialog will close, and the installer program will silently install and register the Satellite Forms redistributable components on the end user's PC without any prompting or interface.

4 Repeat step 1 on each user computer.

Your end user's computers now have the necessary components to communicate with your application.

Tip This silent install procedure could easily be accomplished using a batch file instead of using the Start > Run dialog. You could create a RDKSilentInstall.bat batch file, located in the \Redistribution Kit folder, with these commands:

```
@REM Install the RDK components silently
Setup.exe /S /v/qn
```

You would navigate to the \Redistribution Kit folder and double click on the RDKSilentInstall.bat file to run it, performing the install silently.

Installing your application on the device

This is the final step in deployment of your application. In this step, you will be installing your application on your user's handheld devices.

Procedure Installing your application on the device

1 On the user's computer, navigate to the **Redistribution Kit/Palm** folder you installed in the Interactively install redistributable components on user computers step.

2 Double-click the **Install.bat** file.

A list of HotSync user names appears.

3 Select the correct username and allow the installation to run to completion.

4 At the end of installation, the HotSync Manager restarts automatically.

Note If the user has configured their HotSync Manager to prompt for Exit confirmation before closing, the HotSync Manager will not be able to close and restart automatically. Have the user stop and restart the HotSync Manager manually after installation is complete.

5 After the HotSync Manager has been restarted, verify that your application appears on the handheld device and your application's name appears in the Custom Conduit list of the HotSync Manager on the user's desktop computer.

6 Copy your Hotsync Handler program onto the user's hard drive, or create a batch file to copy the **HotSync Handler** directory automatically onto the user's hard drive.

Installation of your application is complete. Your users can begin using the application.

Working with the Palm OS install utility

The Palm OS install utility is a file-driven generic installer application that is provided to help simplify installing the Satellite Forms engine and application components on your end user's computer. The install utility, **RdkInst.exe**, is located in the following directory:

\Redist\Palm\

The install utility can help you complete the following tasks:

- Install and register the Satellite Forms conduit with the Palm HotSync Manager.
- Uninstall and unregister the Satellite Forms conduit with HotSync Manager (for uninstallation).
- Programmatically restart HotSync Manager so that changes made to the registry take effect.
- Install any number of native files (.PRC or .PDB).

Generally, you will call the install utility from your own installation program to take care of the details of installing and configuring the Satellite Forms conduit, and installing the Satellite Forms engine, your application icons, and your application to a handheld device. Before any files can be installed to the device, the device must first be connected to the HotSync manager, because all files are installed through performing a HotSync.

The install utility is controlled through command line switches and a configuration file. Command line switches are used to request a particular task . For some switches, the configuration file supplies additional data on the task to perform. A usage example and list of command line switches is given below:

Usage: `RDKINST configuration-file [`*`command-line_switch`*`]`

- **crtreg id**: Create a HotSync registry entry with creator id = id. This switch registers the Satellite Forms conduit with the HotSync Manager. It also associates the specified Creator ID with the conduit. This switch is used when deploying to third parties; you should supply the Creator ID supplied to you by PalmSource. The path to the conduit must be specified in the configuration file or with the –**condpath** switch. The HotSync Manager must be restarted after using this command (see **restarths**).
- **remreg id**: Remove HotSync registry entry with creator id = id. (Use to undo –**crtreg** id when uninstalling.) The HotSync Manager must be restarted after using this command (see **restarths**).
- **instprc**: Install .PRC and/or .PDB files. This switch instructs the install utility to install the .PRC/.PDB files listed in the configuration file. The install utility will generate a request to the end user for a HotSync to be performed, and the install utility will continue to run until all files have been installed on the handheld device. When used with **noprcwait**, the install utility will not wait until all have files have been installed to shut down.
- **noprcwait**: Do not wait for HotSync to install .PRC/.PDB files. This switch is used in conjunction with **–instprc** to direct the install utility to prepare the

HotSync manager so that application files will start installing with the next HotSync session, but not to wait for the installation to complete.

- **quiet**: Do not display informational messages. Informational messages are messages that are not due to errors.
- **restarths**: Restart HotSync Manager. This switch is used to stop and restart the HotSync Manager. This step is necessary whenever a change is made to the HotSync Manager's registry entries. The HotSync Manager must be restarted after using the **-crtstdreg**, **-remstdreg**, **-crtreg** and **-remreg** switches.
- **condpath <path>**: Path to the Satellite Forms conduit. If this switch is not specified, the install utility will obtain the condpath setting from the configuration file. Despite the cond*path* name, you should specify the name of the conduit DLL without providing a path, eg. SFrmCn80.DLL. The install utility only needs the condpath value when you use the **-crtstdreg** or **-crtreg** switch.
- **sddi <db-engine>**: Specifies which device database engine to use. If this switch is not specified in the command line or configuration file, the install utility will automatically use the **SDDI_PalmDB.dll** plug-in included with Satellite Forms.
- **crtstdreg**: Create standard HotSync registry entries. This switch registers the Satellite Forms conduit with the HotSync Manager. It also associates the Satellite Forms reserved Creator IDs with the conduit. The path to the conduit must be specified in the configuration file or with the –condpath switch. Since it uses the Satellite Forms Creator IDs, it can only be used with applications deployed internally. Applications deployed externally should use **crtreg id**. The install utility will need the path to the conduit when using this switch (see **condpath <path>**). The HotSync Manager must be restarted after using this command (see **restarths**).
- **remstdreg**: Remove standard HotSync registry entries. (Use to undo **–crtstdreg** when uninstalling.) The HotSync Manager must be restarted after using this command (see **restarths**).
- **? or –help**: Display help information.

Some commonly used switch combinations are shown below:

```
RDKINST install.ini -instprc
```

This switch will install the Satellite Forms engine, your application icons, and the application itself on your end users handheld device.

This set of switches is useful when you are going to install the Satellite Forms engine and your application to many handheld devices. Simply place each target handheld device in the cradle and run the install utility with these switches.

Note The functionality performed with these switches can be achieved directly from your PC application through methods of the ActiveX control. You can install PRC and PDB files with InstallPrcFileToPalmPilot.

```
RDKINST install.ini -condpath SFRMCN80.dll -crtreg xxxx -restarths
```

After you install all the redistributable Satellite Forms components on the end user's computer, run the install utility with these switches to register the conduit with the HotSync Manager. In addition, the creator ID "*xxxx*" of your application will be associated with the conduit and the HotSync Manager will be restarted so the changes take effect. Typically, this procedure is performed one time, during installation of your application.

Deploying Pocket PC applications

This section lists and describes the steps to deploy a Pocket PC application. After you have completed the steps in this section, your application will be distributed to your end users and your development cycle will be complete.

Integrated Runtime

Starting with Satellite Forms Version 8, a new integrated runtime engine feature has been added to improve application deployment. This feature works by combining your application launcher icon with the Satellite Forms runtime engine executable, to create a customized runtime branded specifically for your application. Previously, the launcher icon app was a small “loader” customized with your icon that would load the standard RDK runtime engine to run your application. Now the launcher icon and runtime engine are integrated together into one.

The integrated runtime improves app deployment by having one less file to install as a part of your package. More importantly, it makes your deployed app more robust by reducing the chance that installing another Satellite Forms application might affect your installed application by overwriting the runtime engine with a different version. For the Windows Mobile/Pocket PC platform, the EXE runtime engine and your icon are combined into your application EXE file. The runtime engine DLL file is still required however, and is not integrated with the icon file.

In order to have MobileApp Designer generate an integrated runtime engine for your application, you must specify a proper launcher icon in the Project Properties.

Create and assign application icons

You can specify large and small icons for your application. The icons for your project will appear as small images that represent your application to end users on their devices. This section includes information on creating and assigning the icons to your application. The Mobile AppDesigner program will also generate a launcher icon for your application automatically, discussed below.

For an application with a complete set of icons, create the following:

- Large icon: 32x32 pixels. Utilized for the Large Icon view setting on the handheld device.
- Small icon: 16x16 pixels. Utilized for the List View setting on the handheld device and also in the Start menu.

If you want MobileApp Designer to generate a custom launcher for your application using your own icon, you have two options. You can use a set of small and large color bitmap files, or you can use a multi-image Winodws ICO icon file. *The option to use bitmap files instead of an ICO file is new with Satellite Forms 8.*

To use multiple bitmap files, the large icon bitmap must be 32x32 pixels in size, and 16 or 256 colors. The small bitmap should be 16x16 pixels, 16 or 256 colors. The small icon needs to be named the same as the large icon, with a -Small filename suffix. When you specify the large icon filename, MobileApp Designer will look for the -Small file automatically in the same folder.

To use a multi-image Windows ICO icon file instead of bitmaps, follow the steps below:

Guidelines to follow when creating your icons:

- All files must be in Windows icon format (icon files have a **.ico** extension).
- Icons should be named *YourAppName*.ico, where *YourAppName* is the name of your application project (*.**sfa**) file. *Note that this can be different than the Name of Application in the project properties.*

For general tips on creating icons using a common software program, see the list in the Create and assign application icons section of this chapter for Palm OS.

Create a launcher icon

Mobile AppDesigner can create an custom launcher for your application. Follow the steps below to create a launcher icon:

Procedure Creating a launcher icon

1. Create a multi-image Windows icon file (icon files have a .**ico** extension) with two .bmp files, one large (32x32) and one small (16x16).
2. Name the .ico file *YourAppName*.ico where *YourAppName* is the name of your application project (*.**sfa**) file. *Note that this can be different than the Name of Application in the project properties.*
3. Create an Images directory at the same level as your application source file (the application source file will have a **.sfa** extension). Place *YourAppName*.ico in the Images directory.

Tip Placing all the other images associated with your application, such as .bmp files, in the Images directory will make your application easier to manage.

4. Rebuild your application.

   Your launcher icon is now associated with your application. A launcher file is created in the AppPkg target subfolder named *NameOfApplication.*EXE along with the *NameOfApplication*.PDA file.

Tip An example of a completed launcher icon is also included with the Work Order sample in \Samples\Projects\Work Order.

Create an application shortcut

A shortcut (.Lnk) file for your application EXE is also created in the AppPkg target subfolder, named *NameOfApplication*.Lnk. This shortcut link file aids deployment by making it easy to create an icon in the Programs folder on the Pocket PC that is a shortcut to your application EXE file in your application folder. The user can just tap on your icon in the Programs folder to launch your application. The shortcut .Lnk file is created using the \My Documents\NameOfApplication\*NameOfApplication*.EXE program path. If you deploy your application to a different folder, you'll need to edit the contents of the shortcut .Lnk file using any standard text editor, to change the program path. See the KnowledgeBase article "How To Create a shortcut to your

PocketPC application" for more details on the .Lnk file format. The *NameOfApplication*.Lnk file should be placed into the \Windows\Start Menu\Programs folder on the Pocket PC. Note that the ".Lnk" filename suffix may not be visible in Windows Explorer on the PC, as that is a reserved filename extension that Windows usually hides on the PC.

Prepare the ActiveSync event handler application

Data synchronization between your application and the user's desktop computer can be handled by the Satellite Forms ActiveSync Control, which you can utilize in your desktop application to transfer data back and forth when an ActiveSync is carried out. This is conceptually similar to the use of a Hotsync Handler application for PalmOS devices, but the methods and properties of the ActiveSync control are different than the Hotsync control. Please refer to the Satellite Forms ActiveSync ActiveX control for Pocket PC on page 214. Data synchronization can also be handled without using the ActiveSync ActiveX control, by using the CeRemote.dll instead. Refer to ActiveSync without ActiveX on page 219 for the details on that method.

The instructions below are relevant whether you are using the Satellite Forms ActiveSync Control or the non-ActiveX DLL method.

Set up the redistribution kit

In this step, you will create a folder and fill it with files copied over from the Satellite Forms program files directory, and your project folder. When you are finished, the new folder will contain all the redistributable components for your application.

⚠ Caution Follow the instructions below carefully; the directory structure and naming must match the directions given exactly for your installation to proceed smoothly.

Procedure Preparing the redistribution kit.

1 Create a new, stand-alone directory named **Redistribution Kit**.

2 Create a new directory beneath the Redistribution Kit directory named **PocketPC**.

The directory structure has now been created.

Tip Navigating to the Satellite Forms Program Files directory now will make the rest of the steps in this procedure easier, as you copy and paste files between the Satellite Forms directories and your new redistribution directories. All pathnames given below are from the root of the Satellite Forms directory.

The default location of the Satellite Forms program is C:\Satellite Forms 8\.

3 Copy the **SFrmAxPPC_Install.exe** ActiveSync control installer file from the following location:

\Redist\PocketPC

Paste the contents into the new **Redistribution Kit** directory created in Step 1.

4 Copy all files and subdirectories from the following location:

\Redist\Common\Runtime Installer\Disk

Paste the contents into the new **Redistribution Kit** directory created in Step 1.

Note These redistributable components are required for synchronizing Satellite Forms PDB databases on both the Palm OS and Pocket PC platforms.

5 Copy all files from the following location in your project folder:

\AppPkg\PocketPC (mmnn) Where **mm** is the major version number of your application, **nn** is the minor version number. *If you have used a different platform target instead of PocketPC, then copy the files from that AppPkg subfolder instead of the PocketPC (mmnn) folder*.

Paste the files into the new **Redistribution Kit\PocketPC** directory created in Step 2.

Note that starting with Satellite Forms 8, if your application uses any SFX extension files, those files will be included in your AppPkg folder along with the other application EXE, PDA and PDB files. The integrated runtime engine EXE will also be placed in the AppPkg folder. *In previous versions the runtime engine and extension SFX files had to be copied seperately.*

6 Navigate to the following location in the Satellite Forms program folder:

\Redist\PocketPC

Copy the files named **CeRDKInst.exe**, **CeRemote.dll**, and **SFConvertPDB.exe** and paste them into the **Redistribution Kit** directory created in Step 1.

7 Create the **Install.ini** file that will install your application in a text editor, and place it in the **Redistribution Kit** folder. The Install.ini file is the configuration file that will control which files are included during installation. An example file of a typical Install.ini file is shown below:

Example 10.4 Sample Pocket PC Install.ini file

```
;;
; Installer Section
;
; Key Description
;
; EngWinTitle - Window title when installing engine.
; EngInstruct - User instructions that appear when installing application.
;
[Installer]
EngWinTitle = Your Application Installer
;
; FilesX Sections
;
; One or more group(s) of files to be copied to the device.
;
; Note: Create one key per file to install. All files must be listed with
; keys of the form FileX or TableX where X are consecutive integers starting
; with zero.
;
; Key Description
;
; DevDir - Device destination directory to put files during copy. If the
; directory string does not contain a leading backslash '\' then the
; destination directory will be relative to '\My Documents'. If the
; directory on the device does not exist, the directory will be created.
```

```
: FileX - File to be copied from the desktop to the device. These files are
; copied as is and do not get converted.
; X starts at 0.
;
[Files0]
DevDir = \Program Files\Your Application Name
File0 = PocketPC\YourAppName.EXE
File1 = PocketPC\YourAppName.PDA
File2 = PocketPC\DvSDDI_PPCPDB.DLL
File3 = PocketPC\SFE_Extension1.sfx
File4 = PocketPC\SFE_Extension2.sfx
File5 = PocketPC\ECRIDmmnn_TABLE1.PDB
File6 = PocketPC\ECRIDmmnn_TABLE2.PDB
…
FileN = PocketPC\ECRIDmmnn_TABLEn.PDB
;
;Files1 Section - shortcut file for Your Application Name
;installed to \Windows\Start Menu\Programs
[Files1]
DevDir = \Windows\Start Menu\Programs
File0 = PocketPC\Your Application Name.Lnk
```

As before, *CRID* in the ini file above would be replaced with your actual Creator ID, *mm* would be replaced with your actual major version number, and *nn* would be replaced with your actual minor version number.

Note For detailed information on CeRDKInst utility, refer to Working with the Pocket PC install utility on page 252. For a sample Install.ini and Install.bat, refer to the Redist Work Order sample in the \Satellite Forms 8\Samples\Redist\Work Order directory.

Tip The Install.ini sample demonstrates the inclusion of a program shortcut link (.Lnk) file, which creates a shortcut to your handheld application in the Programs folder on the Pocket PC. This is a very useful addition as it makes it easier for the end user to launch your application on the handheld. See the KnowledgeBase article “How To Create a shortcut to your PocketPC application” for more information. Note that MobileApp Designer creates the shortcut file using “\My Documents” as the base folder for your application on the PDA: if you deploy to another base folder like the “\Program Files” folder shown in the example above, you’ll need to edit the shortcut .Lnk file accordingly.

8 Create the **Install.bat** file that will install your application (including the Satellite Forms runtime engine) to the handheld, and place it in the **Redistribution Kit** folder. Your Install.bat file could resemble the following example:

Example 10.5 Sample Pocket PC Install.bat file

```
@REM Install PDA application using CeRDKInst
CeRDKinst install.ini -instfiles
```

9 Your application, consisting of the new Redistribution Kit directory created in Step 1 and all files and subdirectories you added within the new directory, is now complete and ready to distribute to your end users. Prepare your application for distribution by completing one of the following steps:

- **Network Installation**: Copy the Redistribution Kit directory and all contents onto a network hard drive.
- **CD Installation**:Copy the Redistribution Kit directory and all contents onto a CD-ROM.

Note The above instructions do not include the installation of a desktop application that you will synchronize your handheld data with (via the SatForms ActiveSync control). You must use an appropriate installation method for your desktop sync application, in addition to these instructions for installing the Satellite Forms components.

Distributing the application

This section provides step-by-step instructions on distributing your application to your users.

Caution A well-planned deployment should include testing the distribution steps below yourself, with a clean computer and device, before beginning distribution to a wider user audience. At a minimum, you should confirm that your application installs successfully on both the computer and the device, and that ActiveSync functions correctly.

Verify end user system requirements

Your Satellite Forms application requires a variety of different minimum system requirements to function smoothly. Verify that all system requirements given below have been met by your user’s equipment before proceeding with installation. If you cannot verify the system requirements for your end users, be sure that you transfer the system requirements listed below to the general redistribution readme (see the readme step in Preparing the redistribution kit.)

Desktop computer requirements

Satellite Forms has been tested with the following operating systems:

- Microsoft Windows Vista Business edition
- Microsoft Windows XP Professional edition
- Microsoft Windows 2000 Professional edition
- Microsoft Windows NT 4.0 Workstation with SP 6
- Windows ME
- Windows 98 Second Edition

Handheld device requirements

The following list describes the recommended hardware and software minimum requirements for the handheld devices on which the Satellite Forms runtime engine is installed.

- Pocket PC 2002, 2003 or 2003SE
- Windows Mobile 5.x for Pocket PC, including Phone Edition
- Windows Mobile 6.x Classic or Professional edition

- 10MB available main memory
- ActiveSync 3.5 or higher

Installing redistributable components

You end users will need to transfer information between your application on their handheld device and their computer. The install process for your application includes installing several components on your end user's desktop computers. This step must be completed before installing the application itself on your user's handheld devices.

For Pocket PC applications, the redistributable components required include the **Satellite Forms ActiveSync Control** and the **Satellite Forms Runtime for PCs**. In previous releases this component package was named the Satellite Forms Runtime for *Palm*, but with the adoption of the Palm DB (PDB) handheld database format for the Pocket PC platform the runtime package has been renamed.

The Satellite Forms ActiveSync Control and Satellite Forms Runtime for PCs can be installed on the end user's computer *interactively* (in which the end user sees the SatForms installer prompts and must click several buttons to either complete or cancel the installation) or *silently* (in which the components are installed and registered without presenting any prompts or interface to the end user). Select the interactive or silent install method according to your needs, described below.

Installing redistributable components *interactively*

Complete the following procedure on each user's computer.

Procedure Interactively install redistributable components on user computers

1 Complete the option that applies to your distribution format:

- **Network install**: Navigate to the Redistribution Kit directory and double-click **Setup.exe**.
- **CD-ROM install**: Double-click **Setup.exe** at the root of the CD.
- **Custom installer**: Run your installer.

The installer program starts. Follow the directions of the installer to complete installation of the Satellite Forms Runtime for PCs components.

2 Next, double-click on **SFrmAxPPC_Install.exe** to install the Satellite Forms ActiveSync Control. Follow the directions of the installer to complete that installation.

3 Repeat steps 1 and 2 on each user computer.

Your end user's computers now have the necessary components to communicate with your application.

Installing redistributable components *silently*

Complete the following procedure on each user's computer.

Procedure Silently install redistributable components on user computers

1 Use the Windows Start > Run function to run the setup program with some commandline switches that specify silent install mode. Click on Start > Run and then click the Browse button which opens a standard file selection dialog. Navigate

to the Redistribution Kit directory on your installation media (eg. CD or network folder), click on **Setup.exe**, then click on Open. That will return you to the Run dialog with the full path to the Setup.exe file shown in quotes.

2 Move the cursor to the end of that line without overwriting the path to Setup.exe, and add the commandline switches **`/S /v/qn`** to specify the silent mode, leaving a single space after the end quote, before the switches. You must enter these commandline switches *exactly as shown*: the quoting and spacing matters.

An example commandline shown in the Run dialog, assuming a CD install media using drive letter X: would be:

```
"X:\Redistribution Kit\Setup.exe" /S /v/qn
```

3 Click on OK. The Run dialog will close, and the installer program will silently install and register the Satellite Forms redistributable components on the end user's PC without any prompting or interface.

4 Next, install the Satellite Forms ActiveSync Control silently using the same Start > Run technique. Select **SFrmAxPPC_Install.exe** from the file browse dialog, and add the commandline switch **`/VERYSILENT`** to the end, again outside of the end quote. An example commandline shown in the Run dialog, assuming a CD install media using drive letter X: would be:

```
"X:\Redistribution Kit\SFrmAxPPC_Install.exe" /VERYSILENT
```

5 Repeat steps 1-4 on each user computer.

Your end user's computers now have the necessary components to communicate with your application.

Tip This silent install procedure could easily be accomplished using a batch file instead of using the Start > Run dialog. You could create a RDKSilentInstall.bat batch file, located in the \Redistribution Kit folder, with these commands:

```
@REM Install the RDK components silently
Setup.exe /S /v/qn
@REM Install the ActiveSync Control silently
SFrmAxPPC_Install.exe /VERYSILENT
```

You would navigate to the \Redistribution Kit folder and double click on the RDKSilentInstall.bat file to run it, performing the install silently.

Installing your application on the device

This is the final step in deployment of your application. In this step, you will be installing your application on your user's handheld devices.

Procedure Installing your application on a Pocket PC device

1 With the handheld device in the cradle, double-click the **Install.bat** file from your installation media.

2 Follow the installer directions to complete installation.

3 After installation is complete, verify that your application appears on the handheld device.

Installation of your application is complete. Your user can begin using the application.

Working with the Pocket PC install utility

The Pocket PC install utility is a file-driven generic installer application that is provided to help simplify installing the Satellite Forms engine and application components on your end user's computer. The install utility, **CeRdkInst.exe**, is located in the following directory:

\Redist\PocketPC\

Generally, you will call the install utility from your own installation program to take care of the details of installing the Satellite Forms engine and your application to a handheld device.

Install utility command line switches

The install utility is controlled through command line switches and a configuration file. Command line switches are used to request a particular task . For some switches, the configuration file supplies additional data on the task to perform. A usage example and list of command line switches is given below:

Usage: `CERDKINST configuration-file [`*`command-line_switches`*`]`

- **instfiles**: Install files listed in the configuration file to the device. (See below for the format of the configuration file.)
- **? or –help**: Display help information.

Some typical switch combinations used with the install utility are shown below:

```
CERDKINST install.ini -instfiles
```

This combination of switches will install the any number of files, such as the Satellite Forms engine and your application, on an end user's handheld device. This set of switches is useful when you are going to install your application on many handheld devices. Simply place each target handheld device in the cradle and run the install utility with the above switches.

Working with the install utility configuration file

The install utility configuration file guides many of the tasks performed by the install utility. The configuration file is a text file that uses the Windows .INI file format; it can be created with any text editor. Command line switches passed to the install utility control which of the sections of the configuration file will be used.

Many configuration files are very simple. A sample configuration file and an explanation of elements in the file is given below.

Example 10.6 Sample Pocket PC configuration file

```
;
;Installer Section
[Installer]
EngWinTitle = Sample Installer
```

```
;
;FilesX Section
[Files0]
DevDir = Filter Test
File0 = D:\Apps\Test\FilterTest.EXE
File1 = D:\Apps\Test\FilterTest.PDA
File2 = D:\Apps\Test\ESMSF0000_TABLE1.PDB
;
```

Typically, an INI file is divided into sections. Each section starts with a word in brackets and the section continues to the beginning of the next section or the end of the file. Within each section are keys and values in the form **key = value**. Lines beginning with a semicolon (**;**) are comments and are ignored. The sections in the sample file are described below:

- **[Installer]**

  ```
  EngWinTitle = Sample Installer
  ```

  The [Installer] section controls various settings that specify the dialog boxes the installer will display to the end user.

  The **EngWinTitle** key controls what the install utility shows to the user when it is being used to install files. As seen above, the **EngWinTitle** specifies the title (“Sample Installer”) the installer window will have.

  ```
  EngInstruct = Special instructions can go here
  ```

  The **EngInstruct** key controls what optional text the install utility shows to the user when it is being used to install files. As seen above, the **EngInstruct** specifies the text (“Special instructions can go here”) instead of using the default text, which informs the user to click on the Install button to install the files, or the Cancel button to abort the installation.

  The list of files that should be installed is specified in the next section, [FilesX].

- **[FilesX]**

  ```
  [Files0]
  DevDir = Filter Test
  File0 = D:\Apps\Test\FilterTest.EXE
  File1 = D:\Apps\Test\FilterTest.PDA
  File2 = D:\Apps\Test\ESMSF0000_TABLE1.PDB
  ```

  The [FilesX] section controls which device files will be installed when you use the **–instfiles** switch. Files are grouped together by section names identified with consecutive numbers starting from zero (the **X** in [FilesX]). Each group of files can be installed to a different directory on a handheld device. An explanation of each key in the [FilesX] section is given below:

  - **DevDir**: This key controls which directory the files listed in the file group will be installed to on the handheld device. The DevDir value has different meanings based on the existence and location of the backslash character (‘\’) than the OS of the handheld device uses to delimit the directory levels.

    A backslash as the first character of the value specifies a directory starting at the root of the handheld device. If the value does not start with a backslash then the directory is a subdirectory of the **My Documents** directory. The directory value can contain zero or more backslash characters to represent multiple levels of directories. If the directory does not exist on the handheld device, the install utility will attempt to create the directories as needed.

Creating a custom installer for Palm or Pocket PC applications

You may want to create an installer for your application instead of using the generic version that ships with Satellite Forms. This section contains guidelines for creating a custom installer to distribute the necessary components of your application.

File placement and registration

Your custom installer must distribute files from the RDK to specific locations on your end user's computers. It is recommended that your installer places the files in the order given below.

⚠ Caution Each RDK file you distribute must be marked "shared". This is necessary because other Satellite Forms applications may utilize the same files. If a file is not marked "shared", important files can be removed when an application is uninstalled.

Procedure Adding required functionality to your custom installer

Palm OS only

1 This step is required for Palm OS device applications. Continue to step 2 if your application is for Pocket PC devices.

Your installer must install and register the following files in the **System** folder on the target computer:

- All DLL files in \Redist\Palm\WinSys.
- The file **SFrmAx80.ocx** from \Redist\Palm\CommonFilesFolder. This file must be registered.
- The file **InkVwAx.ocx** from \Redist\Palm\CommonFilesFolder. This file is optional; it is required only when Ink controls are used in an application. If you are using Ink controls, you must include and register this file.

Note Installing the optional InkVwAx.ocx file is recommended to ensure compatibility with possible future applications.

Tip A registration utility, named **regsvr32.exe** and located in \Redist\Palm, is included to simplify registering these files.

2 Install and register files in the **Common Files** folder on the target computer, where **Common Files** represents the directory typically located at **C:\Program Files\Common Files**, using the following guidelines:

a Create a directory named **Satellite Forms *X.X*\Bin** in the Common Files folder.

b In the **Satellite Forms X.X\Bin** directory you created, add the **SDDI** and **SFDDB** directories and their contents from the **\Redist\Palm** directory, making sure to mark all DLL files in these directories as shared.

c Add three registry keys as follows:

- **Path**: Path is a string value which must display the proper path to the **Common Files** directory that contains the **Satellite Forms X.X\Bin** directory. The Path registry key is located in **HKEY_LOCAL_MACHINE\SOFTWARE\Thacker\Satellite Forms *X.X\X.X***, where ***X.X*** represents your Satellite Forms version. An example path registry value would be:

```
C:\Program Files\Common Files\Satellite Forms 8\
```

- **Version**: Version is a DWORD value that contains the current version number. The Version registry key is located in **HKEY_LOCAL_MACHINE\SOFTWARE\Thacker\Satellite Forms *X.X*\CurrentVersion**, where ***X.X*** represents your version of Satellite Forms.The current value should be, in hexadecimal, **0x00000800**.
- **Keyname**: Keyname is a string value which must contain the current SatForms version number in string format. The Keyname registry key is located in **HKEY_LOCAL_MACHINE\SOFTWARE\Thacker\Satellite Forms *X.X*\CurrentVersion**, where ***X.X*** represents your version of Satellite Forms.The current value should be, in string format, **8.0**.

Tip Steps 1 and 2 above can be completed automatically with the merge module included in the RDK that uses the Microsoft Windows Installer. The merge module, named **SatFormsRedist.msm**, is located at \Redist\Palm\Merge Modules. It contains all the files and setup information mentioned above. Include this module in your Windows Installer project for Palm OS and Pocket PC applications. Confirm that steps 1 and 2 above have been completed after your installer is run to ensure the merge module was utilized correctly.

Palm OS only

3 This step is required for Palm OS device applications. Continue to step 4 if your application is for Pocket PC devices.

For Palm OS applications, a third registry key is needed for conduit information. This will contain all the information the HotSync application will need to synchronize with your application's files.

You will need to specify the following:

- Name of the current Satellite Forms conduit. This conduit is located in \Redist\Palm\WinSys. The name should be similiar to the following:

  SFrmCn*XX*.DLL

  where ***XX*** represents your Satellite Forms major version number (eg. 80).
- The unique Creator ID of your application.

Tip The easiest way to install the conduit properly is include the generic install utility within your custom installer program using the command line options. See theWorking with the Palm OS install utility or Working with the Pocket PC install utility sections in this chapter for more information on the generic install utilities.

Caution If you choose to utilize the generic install utility within your custom installer, your installer must complete the DLL installation and registry steps above before starting the generic installer. Be sure to place the generic installer at the end of your custom install process.

4 After you have completed your custom installer, continue with the deployment process in the Deploying Palm OS applications and Deploying Pocket PC applications sections in this chapter.

Chapter 11
Satellite Forms Scripting Language Reference

This chapter describes the Satellite Forms object model, provides an example of creating a Satellite Forms script, and provides a complete reference to the scripting language.

Overview of the Satellite Forms scripting language

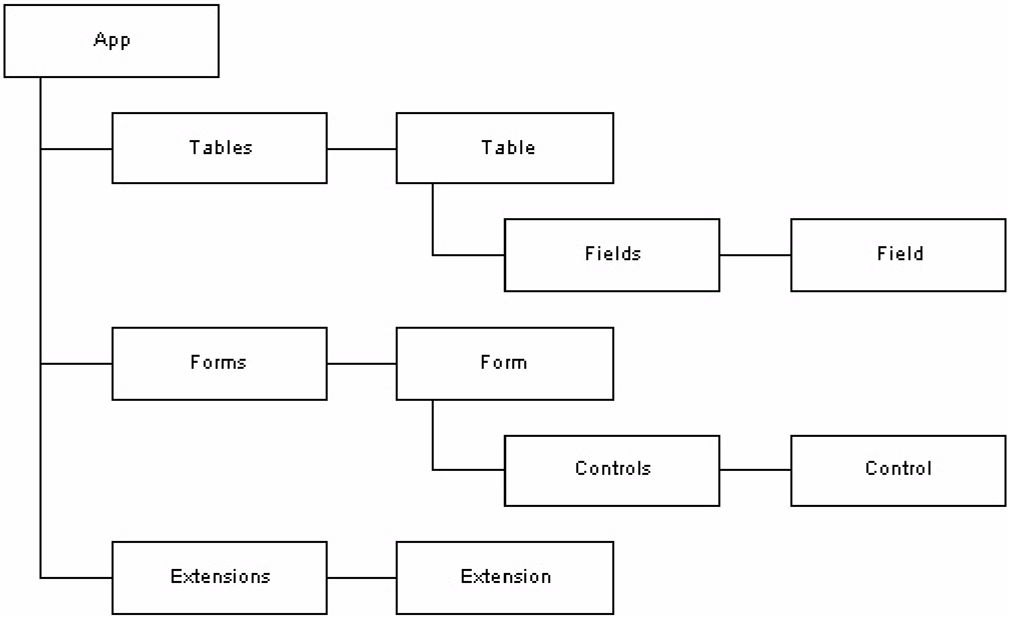
The Satellite Forms scripting language, which is based on Visual Basic syntax, allows you to enhance the functionality of your applications. Scripting complements the built-in Satellite Forms features with local and global variables, mathematical operations, conditional logic, loops, and user-interface functions.

Satellite Forms is based on an object model that provides event-driven scripting. Satellite Forms' objects include the application, forms, controls, tables, fields, and extensions. Objects have properties, methods, and generate events. When an object generates an event, the script associated with that event is executed. Your scripts can take advantage of various object methods and properties as well as many language keyword operators, enabling you to create very versatile applications.

The Satellite Forms object model

The Satellite Forms object model, which is the hierarchy of Satellite Forms objects, is shown in the following figure:

Figure 11.1 Satellite Forms object model



Object properties, methods, and events

Objects have properties, methods, and generate events. The following sections are organized by object, listing and describing the properties, methods, and events for each object. See Satellite Forms scripting language reference on page 314 for detailed syntax and usage information for all properties and methods.

App object properties, methods, and events

The App object is the current Satellite Forms application.

Table 11.1 App object property

Property	Description
None	

Table 11.2 App object methods

Method	Description
Beep	Issues a beep from the handheld device speaker.
DateToSysDate	Converts a user-readable date to days since 1904.
Delay	Waits a specified number of milliseconds.
FormatNumber	Formats a number to a specified number of decimal places.
Function	User defined global function that can take optional parameters, perform your user-defined statements, and return a result.
GetAppCreator	Returns a string containing the application's 4-character creatorID.
GetAppName	Returns a string containing the name of the application as defined in the project properties.
GetAppPath	Returns a string containing the folder path in which the application resides on the PocketPC device.
GetAppVersion	Returns a string containing the application major and minor version numbers as defined in the project properties.
GetEngineVersion	Returns the version number of the runtime engine as a string.
GetLastKey	Returns the last key that pressed.
GetPenStatus	Returns whether or not the pen is touching the screen and if so, the pen's x, y coordinates.
GetSysDate	Returns the current sync state of the specified sync step in a server application.
GetSysDate	Returns days since January 1, 1904.
GetSysTime	Returns seconds since 12 a.m., January 1, 1904.
GetTickCount	Returns the number of timer ticks since the handheld device was turned on.
GetTickFrequency	Returns the frequency of the timer ticks.
GetUserID	Returns the Satellite Forms-assigned unique user ID of the handheld device.
GetUserName	Returns the Satellite Forms-assigned unique user name of the handheld device.
KillTimer	Turns off the timer.
MsgBox	Displays a dialog box with an OK button.
Prompt	Displays a dialog box with OK and Cancel buttons.

Table 11.2 App object methods *(Continued)*

PromptCustom	Displays a dialog box with customizable title and buttons.
Quit	Quit the application and return to the app launcher.
RemoveAllFilters	Removes (clears) all active table filters to make all records visible.
SetDelayToChangeEvent	Sets the delay between when a Graffiti stroke is entered and the firing of the AfterChange event.
SetTimer	Turns on the timer.
Sub	User defined global function that can take optional parameters and perform your user-defined statements. It does not returns a result.
SysDateToDate	Converts days since January 1, 1904 to a user-readable date.
SysTimeToTime	Converts seconds since midnight to a user-readable time.
TimeToSysTime	Converts user-readable time to seconds since midnight.
Tone	Plays a tone of specified frequency, duration, and amplitude.

Table 11.3 App object events

Event	Description
AfterAppStart	Event occurs when the application starts, prior to displaying the first form.
BeforeAppEnd	Event occurs immediately prior to the application closing.

Control object properties, methods, and events

A Control object is a Satellite Forms user-interface control in the current application.

Table 11.4 Control object properties

Property	Description
Caption	Label associated with a control.
Data (default)	Data displayed in the control.
Font	Font used by the control.
Index	Read-only. Index of the control used by Satellite Forms extensions.
ReadOnly	Read-only attribute of the control.
Underline	Underline attribute of the control.
Visible	Makes the control is visible or invisible.

Table 11.5 Control object methods

Method	Description
ExecAction	Executes the action associated with the control.
GetPosition	Retrieves the location and size of the control on the form.
GetSelection	Retrieves the start and end offsets of the highlighted text in the control. Available only for Paragraph and Edit controls.
Popup	Pops up a droplist list, or an autokeyboard for edit & paragraph controls.
SetFocus	Puts cursor in the control.
SetPosition	Modifies the position and size of the control on the form.
SetSelection	Highlights text in the control. Available only for Paragraph and Edit controls.
Scroll	Scrolls the control up or down. Available only for the Paragraph control.

Table 11.6 Control object events

Event	Description
OnClick	Occurs when the user taps a control.

Controls collection properties, methods, and events

A controls collection is the collection of Satellite Forms user-interface controls on a specific form of the current application.

Table 11.7 Control collection properties

Properties	Description
Count	The number of controls in the form.

Table 11.8 Control collection methods

Method	Description
None	

Table 11.9 Control collection events

Event	Description
None	

Extension object properties, methods, and events

An extension object is a Satellite Forms SFX plug-in or SFX Custom control in the current application.

Table 11.10 Extension object properties

Property	Description
Index (default)	Index of the extension.

Table 11.11 Extension object methods

Method	Description
User-defined	See the Help information for the specific extension.

Table 11.12 Extension object events

Event	Description
OnClick	May occur when the user taps a Satellite Forms SFX control. See the Help information for the specific extension for more information.

Extensions collection properties, methods, and events

An Extensions collection is the collection of extensions in a specific application.

Table 11.13 Extensions collection properties

Properties	Description
Count	The number of extensions in the application.

Table 11.14 Extensions collection methods

Method	Description
None	

Table 11.15 Extensions collection events

Event	Description
None	

Field object properties, methods, and events

A Field object is a cell in a specific table in an application. Field objects have the same name as their corresponding columns and always have the context of the current record.

Table 11.16 Field object properties

Properties	Description
Data (default)	Data contained in the field.
Index	Read-only. Index of the field. Used by Satellite Forms extensions.

Table 11.17 Field object method

Method	Description
None	

Table 11.18 Field object events

Event	Description
None	

Fields collection properties, methods, and events

A Field collection is the collection of fields in a specific table.

Table 11.19 Field collection properties

Properties	Description
Count	The number of columns in the table.

Table 11.20 Field collection method

Method	Description
None	

Table 11.21 Field collection events

Event	Description
None	

Form object properties, methods, and events

A Form object is a Satellite Forms form in a specific application.

Table 11.22 Form object properties

Properties	Description
CanClose	CanClose property of the form. **Pocket PC platform only.**
CurrentPage	Zero-based page number of the current page.
CurrentRecord	Zero-based record number of the current record.
Index (default)	Read-only. The index of the form. Used by Satellite Forms extensions.

Table 11.23 Form object methods

Method	Description
GetFocus	Returns the index of the control that has the focus using %Fnnn.Cnnn format or “” if no control has focus. The F in this format represents form, the C represents control, and the nnn represents the index.
MoveFirst	Moves to the first record in the form’s linked table.
MoveLast	Moves to the last record in the form’s linked table.
MoveNext	Moves to the next record in the form’s linked table.
MoveNextPage	Moves to the next page of a form.

Table 11.23 Form object methods *(Continued)*

MovePrevious	Moves to the previous record in the form's linked table.
MovePreviousPage	Moves to the previous page of a form.
PreviousForm	Returns to the previous form.
Refresh	Saves the contents of the form's controls to the underlying table and then calls Requery to reload the data and redraw the screen.
Repaint	Redraws the form and the controls on the screen.
Requery	Reloads controls on the form from the underlying table and then calls Repaint.
Show	Displays the specified form (jumps to the form).

Table 11.24 Form object events

Event	Description
AfterChange	Occurs after data in any field in the form's linked table is changed.
AfterLoad	Occurs after the controls on a form are loaded with data from the form's linked table.
AfterOpen	Occurs after a form is opened.
AfterRecordCreate	Occurs after a new record is created in a form's linked table.
BeforeClose	Occurs immediately before a form is closed.
BeforeRecordDelete	Occurs immediately before a record is deleted from a form's linked table.
OnClick	Occurs when the user taps a control.
OnKey	Occurs when user presses one of the keys or silk-screened buttons, or enters a Graffiti stroke.
OnPenDown	Occurs when the stylus touches the screen.
OnPenUp	Occurs when the stylus is lifted from the screen.
OnTimer	Occurs every period of the timer.
OnValidate	Occurs when a form is validated.

Forms collection properties, methods, and events

The Forms Collection object is the collection of forms in a specific application.

Table 11.25 Form collection properties

Properties	Description
Count	The number of forms in the application.

Table 11.26 From collection methods

Method	Description
None	

Table 11.27 Form collection events

Event	Description
None	

Table object properties, methods, and events

A Table object is a Satellite Forms table in an application.

Table 11.28 Table object properties

Property	Description
Index (default)	Read-only. Index of the table. Used by Satellite Forms extensions.
Count	Read-only. Number of records in the table.
Position	Row number of the current record in the table.
RecordValid	Indicates whether Position specifies a valid record.

Table 11.29 Table object methods

Method	Description
AddFilter	Adds a filter to a table.
Backup	Backs up a table to a specified folder. *New in Satellite Forms 8.*
BinarySearch	Finds an item in a sorted table.
CommitData	Commits (saves) the cached table to storage immediately.
CreateRecord	Creates a record in the table.
DeleteRecord	Deletes a record from the table.
InsertionSort	Sorts the records in the table using the specified column as the key.
Lookup	Finds an item in a table and returns the contents of another column in that matching record. *New in Satellite Forms 8.*
Max	Returns the maximum value of any record in a specified column of the table.
Min	Returns the minimum value of any record in a specified column of a table.

Table 11.29 Table object methods *(Continued)*

Method	**Description**
MoveCurrent	Moves to the table’s current record. Does not change data displayed on the form.
MoveFirst	Moves to the table’s first record. Does not change data displayed on the form.
MoveLast	Moves to the table’s last record. Does not change data displayed on the form.
MoveNext	Moves to the table’s next record. Does not change data displayed on the form.
MovePrevious	Moves to the table’s previous record. Does not change data displayed on the form.
MoveRecord	Moves a record from location in the table and to another location.
QuickSort	Sorts records in the table using a specified column as the sort key.
RemoveFilter	Clears (removes) a filter from a table.
RemoveRecord	Immediately removes a record from the table. Differs from ***DeleteRecord*** which deletes records at the next HotSync.
Search	Finds an item in a table. *New in Satellite Forms 8.*
Sum	Sums the data in a specified column of the table.

Table 11.30 Table object events

Event	**Description**
None	

Tables collection properties, methods, and events

The Tables Collection object is the collection of tables in an application.

Table 11.31 Table collection properties

Property	Description
Count	The number of tables in the application.

Table 11.32 Table collection methods

Method	Description
None	

Table 11.33 Table collection events

Event	Description
None	

SFX plug-in and control properties and methods

SFX plug-ins and controls have properties and methods you can use. This section provides an overview of the properties and methods for many of the SFX plug-ins and controls supplied with Satellite Forms.

Note Not all of the SFX plug-ins and controls included with Satellite Forms are listed here in this documentation. You are encouraged to explore the sample projects included with Satellite Forms to learn about other extensions that are not listed here.

You can learn more about these and other commercially available extensions in the Satellite Forms Solutions Guide, and also by browsing the sample projects included with Satellite Forms.

Note Some SFX plug-ins and controls are available for Pocket PC devices, some are available for the PalmOS platform only, and some are available for both platforms.

SFX plug-in and control extensions included with Satellite Forms

Table 11.34 SFX plug-in and control extensions included with Satellite Forms

Extension	Description
Aceeca IDVERIFI Bar Code extension	Controls the barcode scanner on Aceeca Meazura ruggedized Palm OS scanners.
Battery Info extension	Gets battery & power status, now replaced by SysUtils extension.
Color Graphics extension	Simple color graphic functions like drawing boxes and circles.
Colorizer extension	Customize your app appearance with color forms and controls.

Table 11.34 SFX plug-in and control extensions included with Satellite Forms

Color Slider control	Provides a color slider control for user input and progress feedback.
ConnectionMgr extension	Connect to or disconnect from the Internet on Pocket PC PDAs.
DynamicInputArea control	Custom control that enables you to support the expandable screens that are available on some Palm OS devices.
EditEx extension	Functions to manipulate Edit and Paragraph controls.
FindFiles extension	Functions to find and list information about files and folders.
FormNavHelper extension	Handles the Treo-style focus ring form navigation system on newer Palm handhelds.
GoogleMaps extension	Access Google Maps for Palm OS from your SatForms application.
GPS extension	Easy access to GPS data on Windows Mobile 5 and higher devices.
HyperLink control	Add colored, underlined text links that respond to pen taps.
InkHelper extension	Utility and conversion functions for Satellite Forms Ink fields.
IntermecScan control	Controls the barcode scanner on Intermec Pocket PC scanners.
IntermecScan control	Controls the barcode scanner on Honeywell Pocket PC scanners.
IntermecScan control	Controls the barcode scanner on Datalogic Pocket PC scanners.
JanamUtils extension	Control hardware properties on Janam XP devices.
LaunchReturn extension	Restart your application after the user runs another program.
LaunchURL extension	Launch/view a specified website URL, local html documents and image files in the web browser.
Math extension	Advanced mathematical functions including trigonometry.
Memory extension	Allocate, read, and write memory blocks in your application.
Printer extension	Simple serial printing functions for Palm OS.
Puma Beam DB extension	Adds infrared (IR) data file transmission capability.
Puma Data Manager extension	Access to a selected set of Palm OS Data Manager APIs.
Puma Error Manager extension	Palm OS error code handling.
Puma Resource Manager extension	Access to Palm OS resources and their databases.
Px Screen Tool extension	Functions to switch Palm OS display modes.
Random Number Generator extension	Pseudo-random number generation capabilities.
ScreenSize extension	Get current screen size and orientation on the PocketPC.
Serial Port extension	Serial port send/receive capability for Palm OS and Pocket PC.
ShowImage control	Display image files (JPG/GIF/PNG/BMP) on the form on Pocket PC devices.

Table 11.34 SFX plug-in and control extensions included with Satellite Forms

SocketScan control	Controls a a Socket Mobile barcode reader card or Cordless Hand Scanner.
Square Root extension	Provides a square root function.
Strings extension	Provides numerous handy string manipulation functions.
Symbol Integrated Scanner control	Controls integrated bar code reader on many Symbol and Janam Palm OS scanners, and Symbol Pocket PC handheld scanners.
Symbol MSR control	Controls a Symbol Palm OS magnetic stripe reader.
SysUtils extension	Access to dozens of OS (operating system) utility functions
TCPIP Winsock/Internet extension	Connect to TCP/IP based networks (such as the Internet) to send/receive data over sockets.
UnitechScan control	Controls integrated barcode scanner on Unitech Pocket PCs.
WM5Camera extension	Control an integrated camera on Windows Mobile 5 and higher devices to capture still photos and videos.

A description of the methods and properties of many of the above listed extensions appears below.

Aceeca IDVERIFI Bar Code extension

The Aceeca IDVERIFI Bar Code extension interfaces to an integrated bar code scanner on Aceeca Meazura Palm OS devices. The Aceeca IDVERIFI Bar Code extension is a plug-in and has no properties.

Platform(s): Palm OS only (Aceeca Meazura devices only)

Sample project(s): IDVERIFI Barcode

Table 11.35 Aceeca IDVERIFI Bar Code extension methods

Method	Description
About	Displays information about the extension in a dialog box.
IDV_IsAceecaUnit	Returns true or false if the current device is an Aceeca unit.
IDV_SetBCSType	Sets the installed BCS module type for BCS1 or BCS2.
IDV_EnableScanner	Turns on scanner power, enables UART. Returns true if scanner enabled, false if not enabled.
IDV_DisableScanner	Turns off power to scanner, disables UART. Returns true if scanner disabled, false if not disabled.
IDV_Trigger	Starts scanning if trigOn is true, else stops scanning.
IDV_ScanAvail	Returns # bytes in scanner buffer.
IDV_GetScan	Returns barcode string or times out.
IDV_ResetScanner	Cycles power to scanner engine to force it to be reset.
IDV_SetToFactoryDefaults	Sets scanner engine configuration to factory defaults.
IDV_ChangeSettings	Sends configuration string to scanner.

Table 11.35 Aceeca IDVERIFI Bar Code extension methods *(Continued)*

IDV_GetLibVersion	Returns the scanner library version number.
IDV_SetScanTrigger	Specifies a keycode that will trigger a scan, intended to let you program hard buttons to trigger a scan.
IDV_GetScanTrigger	Returns a specified scan trigger keycode.

Battery Info extension

The Battery Info extension allows you to obtain battery charge level and AC power status in your Satellite Forms application. The Battery Info extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Battery Info

Note These battery functions are also available in the SysUtils extension which has numerous other useful methods. **Use the SysUtils extension instead.**

Table 11.36 Battery Info extension methods

Method	Description
About	Displays information about the extension in a dialog box.
BI_GetBatteryPercent	Returns the current battery charge level as a percent.
BI_GetPluggedIn	Returns whether the PDA is plugged in to power or not.

Color Graphics extension

The Color Graphics extension provides color graphic capabilities to your Satellite Forms applications. The Color Graphics extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Color Graphics, Color Table

Table 11.37 Color Graphics extension methods

Method	Description
About	Displays information about the extension in a dialog box.
CalcTextWidth	Determine the width in pixels a given string requires, using the current font.
DrawBar	Draws a filled bar, as for a bar graph, using the current Pen and Fill colors.
DrawCircle	Draws a circle using the current Pen and Fill colors.
DrawLine	Draws a line using the current Pen color.
DrawRect	Draws a rectangle using the current Pen and Fill colors.
DrawRoundedRect	Draws a rounded rectangle with the specified corner radius using the current Pen and Fill colors.
DrawText	Draws text at the specified point using the current text color.

Table 11.37 Color Graphics extension methods *(Continued)*

EraseBar	Erases a filled bar drawn using the `DrawBar` method.
EraseCircle	Erases a circle drawn using the `DrawCircle` method.
EraseLine	Erases a line drawn using the `DrawLine` method.
EraseRect	Erases a rectangle drawn using the `DrawRect` method.
EraseText	Erases text drawn using the `DrawText` method.
GetColor	Returns the colors currently in use.
GetCurrentRGBColors	Returns a text string containing the current red, green, and blue colors.
GetFillColor	Returns the Fill color.
GetPenColor	Returns the Pen color.
InvertText	Inverts the text at the specified point.
Is16BitCapable	Determines whether the handheld device is capable of displaying 16-bit color.
Is35	Determines whether the handheld device has Palm OS 3.5 or greater.
RestorePrevColor	Restores the previous color scheme.
SetBackColor	Sets the background color to one of the 256 colors in the color table.
SetBackColor16	Sets the 16-bit background color based on the specified RGB values.
SetFillColor	Sets the Fill color to one of the 256 colors in the color table.
SetFont	Sets font style for the `DrawText` method.
SetForeColor	Sets the foreground color to one of the 256 colors in the color table.
SetForeColor16	Sets the 16-bit foreground color based on the specified RGB values.
SetPaintKeyCode	Specifies a virtual keycode to be sent to your app by the extension as a signal to redraw your graphics. Trap this keycode in the OnKey event. Pocket PC only.
SetPattern	Sets the 8 x 8 custom Fill pattern.
SetPenColor	Set the Pen color.
SetTextColor	Sets the text color to one of the 256 colors in the color table.
SetTextColor16	Sets the 16-bit text color based on the specified RGB values.

Colorizer extension

The Colorizer extension is a plugin extension for Palm OS and Pocket PC that enables you to customize the look of your application with color forms and controls.

See the discussion Using color in your application on page 190 for a more in-depth explanation of how to use the Colorize methods to customize your application appearance.

Platform(s): Palm OS and Pocket PC.

Sample project(s): Colorizer

The Colorizer sample project also demonstrates the use of a splash screen bitmap, as well as the integrated runtime engine. If you installed Satellite Forms in the default location, this sample is located at C:\Satellite Forms 8\Samples\Projects\Colorizer\Colorizer.sfa and can also be reached from the Windows Start menu > Programs > Satellite Forms 8 > Samples > Projects > Colorizer. The Colorizer sample project has also been precompiled into a Redistribution sample that is ready to install onto a Palm OS or Windows Mobile handheld, via the Windows Start menu > Programs > Satellite Forms 8 > Samples > Redist > Colorizer.

Table 11.38 Colorizer extension methods

Method	Description
About	Displays information about the extension in a dialog box.
Colorize	(PocketPC only) Set True to use color controls with the colors you have defined, or False to use standard system colors.
ColorizeButton	Set Button controls foreground and background colors.
ColorizeCheckbox	Set Checkbox controls foreground and background colors.
ColorizeDroplist	Set Droplist controls foreground and background colors.
ColorizeEdit	Set Edit controls foreground and background colors.
ColorizeExtra	(PalmOS only) Set extra Palm UI colors that are not handled in any other Colorizer functions.
ColorizeForm	Set Forms background color as hexadecimal RGB value.
ColorizeInk	Set Ink controls foreground and background colors.
ColorizeListbox	Set Listbox controls foreground and background colors.
ColorizeLookup	Set Lookup controls foreground and background colors.
ColorizeParagraph	Set Paragraph controls foreground and background colors.
ColorizeRadio	Set Radio controls foreground and background colors.
ColorizeText	Set Text controls foreground and background colors.

Color Slider control

The Color Slider control provides a Slider control that uses color graphics for use in your Satellite Forms applications.

Platform(s): Palm OS and Pocket PC

Sample project(s): Color SFX Controls

Table 11.39 Color Slider control properties

Property	Description
BACKCOLOR	Specifies the background color.

Table 11.39 Color Slider control properties *(Continued)*

BEEP	Specifies whether the control should beep when the user interacts with it. If set 0, no beep occurs. If set to 1, a beep sounds when the user first touches the control. The default setting is 0.
DSOURCE	Fields("col_name") specifies that "col_name" (double-quotes required) in the form's linked table that contains the control will be used to load and store the value of the control. The default value is "none".
FASTTRACK	Specifies how often the `OnClick` event fires: • 0 = Every time the pen moves • 1 = After the pen is lifted The default setting is 0.
FORECOLOR	Specifies the foreground color.
INITVAL	Specifies that the initial position of the control's bar should be at a number between MIN and MAX. The default setting is 0.
MIN	Specifies the minimum position of the control. The default setting is 0.
MAX	Specifies the maximum position of the control. The default setting is 100.
PATTERNTYPE	Specifies the style of the bar pattern: • 0 = Transparent • 1 = Alternating dots • 2 = Diagonal lines • 3 = Vertical lines • 4 = Black
STYLE	Specifies the style of slider control: • 0 = Progress Bar: no user interaction) • 1 = Slider with marker: user can slide the marker • 2 = Slider with no marker: user taps the bar to indicate the desired position • 3 = 3D Slider: color handhelds only The default setting is 1.
VISIBLE	Specifies whether the control is visible. The default value is TRUE.

Table 11.40 Color Slider control methods

Method	**Description**
About	Displays information about the control in a dialog box.
FastTrack	Specifies how often the `OnClick` event fires.
GetCurrentColor	Returns the specified color, either the foreground color or the background color depending on the parameter value.
SldGetPosition	Returns the current position of the control.
IsHandheld35	Determines whether the handheld device has Palm OS 3.5 or greater.
RestorePrevColor	Restores the previous color scheme.
SetBackColor	Sets the background color.

Table 11.40 Color Slider control methods *(Continued)*

SetForeColor	Sets the foreground color.
SldSetMinMax	Sets the minimum and maximum values.
SldSetPosition	Sets the position of the control.
SetVisible	Makes the control visible or invisible.

ConnectionMgr extension

The ConnectionMgr extension is a plugin extension for Pocket PC that enables your application to initiate a dialup connection to the Internet. This is useful for TCPIP Winsock functions, HTTP, FTP, etc. on dialup TCPIP connections (eg. modem, EDGE/GPRS, 1xRTT/EVDO, HSDPA, etc.). A disconnect function is also provided.

Platform(s): Pocket PC only. Windows CE devices do not have the ConnectionMgr API libraries, so use CM_HasConnectionMgr first before calling other functions.

Sample project(s): ConnectionMgr

Table 11.41 ConnectionMgr extension methods

Method	Description
About	Displays information about the extension in a dialog box.
CM_Connect	Connect to the internet, using the specified URL to determine the best connection method.
CM_ConnectByIndex	Connect using the specified connection instead of default.
CM_Disconnect	Request the current connection to be disconnected.
CM_GetConnectionName	Generate a list of available connection names.
CM_HasConnectionMgr	Determines if device has the ConnectionMgr API library or not.

DatalogicScan control

The DatalogicScan control interfaces to an integrated bar code scanner on many Datalogic Windows Mobile devices.

Platform(s): Windows Mobile (Datalogic devices only)

Note There are actually two different versions of the SFE_DatalogicScan.sfx extension file provided, and the correct SFX file to use on your handheld device depends on which specific Datalogic scanner model you are using. See the extension help for the DatalogicScan control for the complete explanation. The methods and properties are the same for both versions of the extension.

Sample project(s): DatalogicScan

Table 11.42 DatalogicScan control properties

Property	Description
AFTERSCAN	Specifies a button control to execute immediately after performing the barcode scan, instead of executing the OnClick action of the DatalogicScan control.

Table 11.43 DatalogicScan control methods

Method	Description
About	Displays information about the control in a dialog box.
AddStripChar	Add a char (by ASCII value) to the list of chars that should be stripped from the scanner data output (useful for stripping off CRLF or other).
ClearStripChars	Clear the list of chars that should be stripped from the scanner data output, so no chars are stripped.
DoScan	Triggers a scan as if the hardware scan button was pressed.
GetScanData	Returns the scanned data from the last scan operation.
GetScanDataLen	Returns the length in bytes of the scanned data from the last scan operation.
GetScanOkay	Returns true or false to indicate if the last scan operation was successful.
GetScanType	Returns the symbol code from the last scan operation.
IsDatalogicScanner	Returns whether current device is an Datalogic scanner.
IsScannerEnabled	Returns whether the scanner is enabled (true) or disabled (false).
IsWedgeEnabled	Returns whether the scanner wedge utility is enabled (true) or disabled (false).
ScannerDisable	Disables the scanner.
ScannerEnable	Enables the scanner.
SetAfterScan	Specifies a button control or the DatalogicScan control's OnClick event to execute immediately after the scan event is fired.
SetWedgeEnabled	Sets whether the scanner wedge utility is enabled (true) or disabled (false).

DynamicInputArea control

The DynamicInputArea (DIA) extension is a custom control that enables you to support the expandable screens that are available on some Palm OS devices (such as the Palm Tungsten T3, T5, TX, and LifeDrive). A KnowledgeBase article titled “**How To support Expandable Screens in PalmOS applications**” discusses how to use this DynamicInputArea control in your applications.

Platform(s): Palm OS only

Sample project(s): Dynamic Input Area

Table 11.44 DynamicInputArea control properties

Property	**Description**
DIAPOLICY	How the dynamic input area should be handled on the current form.
DIASTATE	The current setting of the DIA (maximized or minimized).
DIATRIGGERSTATE	The current setting of the input trigger (enabled or disabled).
ORIENTATION	The current screen orientation (landscape or portrait).
ORIENTATIONTRIGGERSTATE	The current status of the orientation trigger (enabled or disabled).

Table 11.45 DynamicInputArea extension methods

Method	**Description**
About	Displays information about the extension in a dialog box.
DIA_DeviceHasDIA	Returns whether the current device has a DIA.
DIA_GetCurrentPinlet	Returns current pinlet (DIA input keyboard).
DIA_GetCurrentPinletMode	Returns current pinlet (DIA input keyboard) mode.
DIA_GetDIAState	Returns current DIA state.
DIA_GetDisplayHeight	Returns current display height in pixels.
DIA_GetDisplayWidth	Returns current display width in pixels.
DIA_GetFormDIAPolicy	Returns current form DIA policy.
DIA_GetOrientation	Returns current display orientation.
DIA_GetOriTriggerState	Returns current orientation trigger state.
DIA_GetTriggerState	Returns current DIA trigger state.
DIA_SetCurrentPinlet	Sets active pinlet (DIA input keyboard).
DIA_SetCurrentPinletMode	Sets current pinlet (DIA input keyboard) mode.
DIA_SetDIAState	Set current DIA state.
DIA_SetFormDIAPolicy	Sets current form DIA policy.
DIA_SetOrientation	Sets current display orientation.
DIA_SetOriTriggerState	Sets current orientation trigger state.
DIA_SetTriggerState	Sets current DIA trigger state.

EditEx extension

The EditEx extension allows you to manipulate Edit and Paragraph controls in Satellite Forms applications. The EditEx extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Find Replace

Table 11.46 EditEx extension methods

Method	Description
EditExAbout	Displays information about the extension in a dialog box.
EditExAppendText	Appends text to the specified control's text.
EditExBackspace	Backspaces one character in the specified control.
EditExDeleteText	Deletes the specified text from the specified control.
EditExGetInsertion	Returns the current insertion position in the specified control.
EditExGetSelectionEnd	Returns the end position of the text selection in the specified control.
EditExGetSelectionStart	Returns the start position of the text selection in the specified control.
EditExInsertText	Inserts the specified text at the current insertion position in the specified control.
EditExInStr	Returns the position of the first occurrence of one string within another.
EditExSetInsertion	Sets the insertion position in the specified control.
EditExSetSelection	Sets the text selection in the specified control.

FindFiles extension

The FindFiles extension provides functions to search for files and folders, and get information about them such as file sizes and dates. This extension makes it easy to generate a list of PDB files in the app's folder, or a list of documents in the My Documents folder, etc.. The FindFiles extension is a plug-in and has no properties. *New in Satellite Forms 8.*

Platform(s): Palm OS and Pocket PC

Sample project(s): FindFiles

Table 11.47 FindFiles extension methods

Method	Description
FF_DeviceHasVFS	[PalmOS only.] Report whether the device has VFS memory card support.
FF_FindClose	Close an open Find operation when done with it.
FF_FindFirstDir	Find first dir in a new search. [For PalmOS this operates on a VFS card volume.]
FF_FindFirstFile	Find first file in a new search.
FF_FindFirstFileVFS	[PalmOS only.] Find first file in a new search on VFS card.
FF_FindNextDir	Find next dir in existing search. [For PalmOS this operates on a VFS card volume.]
FF_FindNextFile	Find next file in existing search.

Table 11.47 FindFiles extension methods *(Continued)*

FF_FindNextFileVFS	[PalmOS only.] Find next file in existing search on VFS card.
FF_GetFileAttr	Get the file attributes of the current matching file.
FF_GetFileCreator	[PalmOS only.] Get the creatorID of the current matching file.
FF_GetFileDateBackedUp	Get the file backed up date of the current matching file.
FF_GetFileDateCreated	Get the file creation date of the current matching file.
FF_GetFileDateModified	Get the file modification date of the current matching file.
FF_GetFileName	Get the name of the current matching file or directory.
FF_GetFileSize	Get the file size of the current matching file.
FF_GetFileType	[PalmOS only.] Get the file type of the current matching file.
FF_GetFileVersion	[PalmOS only.] Get the file version of the current matching file.
FF_GetLastErr	Report the last error code.
FF_GetNextVolRef	[PalmOS only.] Get the first or next VFS volume reference, which identifies a VFS volume for use in other functions.
FF_GetVFSLabel	[PalmOS only.] Return the specified VFS volume label.
FF_GetVFSVolRef	[PalmOS only.] Gets the currently active global VFS volume reference.
FF_SetCaseSensitive	[PalmOS only.] Set the filename matching case sensitivity true or false for a new search.
FF_SetVFSVolRef	[PalmOS only.] Sets the currently active global VFS volume reference.
FF_ShowPrivateVolumes	[PalmOS only.] Specify whether to make hidden internal private volumes visible to the FF_GetNextVolRef function.

FormNavHelper extension

The FormNavHelper extension provides functions for handling the Treo-style focus ring form navigation system on newer Palm handhelds. The FormNavHelper extension is a plug-in and has no properties.

Platform(s): Palm OS only

Sample project(s): Form Nav Helper

Table 11.48 FormNavHelper extension methods

Method	**Description**
About	Displays information about the extension in a dialog box.
GetNavControlFocus	Gets the index of the control on the current form that has navigation focus.
GetNavState	Returns current form navigation state.
HasFormNav	Returns whether this device has form nav support.
SetNavState	Sets current form navigation state.

Table 11.48 FormNavHelper extension methods *(Continued)*

SetNavControlFocus	Sets current form navigation focus on specified form object.

Generic extension

The Generic extension provides a template for writing your own extensions for use in Satellite Forms applications. The Generic extension has no useful methods nor properties, and is intended solely for use as a starting point for extension development.

Platform(s): Palm OS and Pocket PC

Table 11.49 Generic extension method

Method	Description
About	Displays information about the extension in a dialog box.

GoogleMaps extension

The GoogleMaps extension enables you to launch the Google Maps application to find a location, find a business, get directions to a location, or get directions from a location. The GoogleMaps extension is a plug-in and has no properties.

Platform(s): Palm OS only

Sample project(s): GoogleMaps

Table 11.50 GoogleMaps extension methods

Method	Description
About	Displays information about the extension in a dialog box.
GM_LaunchGoogleMaps	Launches GoogleMaps app to find a location/business/directions.

GPS extension

The GPS extension provides easy access to GPS data on Windows Mobile 5 and higher devices, via the Windows Mobile GPS API. The GPS extension is a plug-in and has no properties.

Platform(s): Pocket PC only (Window Mobile 5+ OS version)

Sample project(s): GPS

Table 11.51 GPS extension methods

Method	Description
About	Displays information about the extension in a dialog box.
GPS_CalcDistance	Calculate the distance in metres between two GPS waypoints.
GPS_CloseGPS	Close connection to GPS receiver, powering down GPS receiver if no other tasks are also using it.
GPS_GetPosLatitude	Get Latitude in decimal degrees from GPS position data.
GPS_GetPosLongitude	Get Longitude in decimal degrees from GPS position data.

Table 11.51 GPS extension methods *(Continued)*

GPS_GetPosOther	Get other GPS position data, indicated by other information type.
GPS_GetPosUTCTime	Get UTC Time from GPS position data.
GPS_GetValidFields	Check whether GPS is returning valid position data.
GPS_HasGPSAPI	Check whether the current device includes the WM5 GPS API.
GPS_OpenGPS	Open connection to GPS receiver, powering up GPS receiver if necessary.

HoneywellScan control

The HoneywellScan control interfaces to an integrated bar code scanner on many Honeywell Windows Mobile devices.

Platform(s): Windows Mobile (Honeywell devices only)

Sample project(s): HoneywellScan

Table 11.52 HoneywellScan control properties

Property	Description
AFTERSCAN	Specifies a button control to execute immediately after performing the barcode scan, instead of executing the OnClick action of the HoneywellScan control.
BADREADEVENT	Controls whether or not the scanner OnClick event will be fired on bad read scan attempts, in addition to good scan reads.

Table 11.53 HoneywellScan control methods

Method	Description
About	Displays information about the control in a dialog box.
DoScan	Triggers a scan as if the hardware scan button was pressed.
EnableSymbology	Enable or disable the decoding of a specific symbology (barcode type).
GetScanData	Returns the scanned data from the last scan operation.
GetScanDataLen	Returns the length in bytes of the scanned data from the last scan operation.
GetScanOkay	Returns true or false to indicate if the last scan operation was successful.
GetScanType	Returns the symbol code from the last scan operation.
IsHoneywellScanner	Returns whether current device is an Honeywell scanner.
IsScannerEnabled	Returns whether the scanner is enabled (true) or disabled (false).
ScannerDisable	Disables the scanner.
ScannerEnable	Enables the scanner.
SetAfterScan	Specifies a button control or the HoneywellScan control's OnClick event to execute immediately after the scan event is fired.

Table 11.53 HoneywellScan control methods *(Continued)*

SetBadReadEvent	Specifies whether to fire the scan OnClick event on a bad read attempt.
SetSymbologyDefaults	Set a specific symbology to use standard default decoding parameters.

HyperLink control

The HyperLink control is a custom control that makes it simple to add colored, underlined text links that respond to pen taps, just like a hyperlink in a web browser. The GoogleMaps extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): HyperLink

Table 11.54 HyperLink control properties

Property	Description
Beep	Set the desired beep sound when the HyperLink is tapped.
Color	Set the color the HyperLink is drawn with.
Font	Set the font of the HyperLink text.
Style	Set whether the HyperLink underline is drawn as a solid line (0) or dotted line (1) or no line (2) when the form opens.
Text	Set the HyperLink text (can be blank if desired).
Visible	Set whether or not the HyperLink is displayed when the form opens.

Table 11.55 HyperLink extension methods

Method	Description
GetBeep	Gets the BEEP property of the control.
GetColor	Gets the COLOR value for the control.
GetControlH	Gets the control's height.
GetControlW	Gets the control's width.
GetControlX	Gets the control's top left X location.
GetControlY	Gets the control's top left Y location.
GetFont	Gets the current FONT of the control.
GetGULColor	Gets the GULCOLOR (global underline color) of the control.
GetStyle	Gets the current STYLE of the control.
GetText	Gets the current TEXT of the control.
GetVisible	Gets the current VISIBLE status of the control.
Hide	Sets the Hyperlink control VISIBLE=false and redraws.
SetBeep	Sets the BEEP property of the control.
SetBounds	Sets the bounds (x,y position and width & height) of the control.

Table 11.55 HyperLink extension methods *(Continued)*

SetColor	Sets the COLOR of the control.
SetFont	Sets the FONT property of the control.
SetGULColor	Sets the GULCOLOR (global underline color) of the control.
SetStyle	Sets the STYLE property of the control.
SetText	Sets the TEXT property of the control.
SetVisible	Sets the VISIBLE property of the control and draws or erases.
Show	Sets the Hyperlink control VISIBLE=true and redraws.

InkHelper extension

The InkHelper extension provides utility functions for working with Satellite Forms Ink fields. It enables you to convert the ink data to other formats including a BMP file, for easy integration with other software, on-device printing, etc. The InkHelper extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): InkHelper

Table 11.56 InkHelper extension methods

Method	**Description**
About	Displays information about the extension in a dialog box.
IH_InkFieldToBitmap	Save the contents of an ink field to a BMP file.
IH_InkFieldToHexText	Return the contents of an ink field as hextext.
IH_FileToBinField	Import a file into a (non-ink) binary field.
IH_FileToHexText	Return the contents of a file as hextext.
IH_FileToUUEText	Return the contents of a file as uuencoded text.
IH_DeleteFile	Delete specified file (eg. delete a BMP after you are done with it).
IH_PalmFileSettings	Specify PalmOS file settings used for other functions that access files.
IH_BMPColorSettings	Set foreground (pen) and background colors (8-bit RGB values) for monochrome BMP file.

IntermecScan control

The IntermecScan control interfaces to an integrated bar code scanner on many Intermec Pocket PC devices.

Platform(s): Pocket PC (Intermec devices only)

Sample project(s): IntermecScan

Table 11.57 IntermecScan control properties

Property	Description
AFTERSCAN	Specifies a button control to execute immediately after performing the barcode scan, instead of executing the OnClick action of the IntermecScan control.

Table 11.58 IntermecScan control methods

Method	Description
About	Displays information about the control in a dialog box.
DoScan	Triggers a scan as if the hardware scan button was pressed.
GetScanData	Returns the scanned data from the last scan operation.
GetScanDataLen	Returns the length in bytes of the scanned data from the last scan operation.
GetScanOkay	Returns true or false to indicate if the last scan operation was successful.
GetScanType	Returns the symbol code from the last scan operation.
GetSettings	Returns current scanner filter settings string.
IsIntermecScanner	Returns whether current device is an Intermec scanner.
IsScannerEnabled	Returns whether the scanner is enabled (true) or disabled (false).
ScannerDisable	Disables the scanner.
ScannerEnable	Enables the scanner.
SetAfterScan	Specifies a button control or the IntermecScan control's OnClick event to execute immediately after the scan event is fired.
SetSettings	Sets current scanner filter settings.

JanamUtils extension

The JanamUtils extension provides access to Janam XP20/XP30 hardware utility functions like toggling the keypad backlight, vibrator, LED, 5V Power Out, and Bluetooth power state.. See the JanamUtils sample project for a demonstration.

Platform(s): Janam XP20/XP30 Palm OS scanners only

Sample project(s): JanamUtils

Table 11.59 JanamUtils extension methods

Method	Description
About	Displays information about the extension in a dialog box.
JXP_5VPowerOut	Get or Set the 5V Power Out (used by MSR attachment) setting on Janam XP devices.
JXP_BTState	Get or Set the BT State (whether Bluetooth is enabled or not) setting on Janam XP units.

Table 11.59 JanamUtils extension methods *(Continued)*

JXP_BuzzHiVolume	Get or Set the Buzz Hi Volume setting on Janam XP units.
JXP_GreenLED	Get or Set the Green LED setting on Janam XP units.
JXP_IsJanamXP	Return whether the current device is a Janam XP unit.
JXP_KeyBacklight	Get or Set the Key Backlight setting on Janam XP devices.
JXP_RedLED	Get or Set the Red LED setting on Janam XP units.
JXP_Vibrator	Get or Set the Vibrator setting on Janam XP handhelds.

LaunchReturn extension

The LaunchReturn extension provides functions that help you restart your application after the user leaves your app to run another program. LaunchReturn can be instructed to 'listen' for the starting and stopping of another app, and can relaunch your app when the other app is closed. LaunchReturn is only functional on Palm OS 5.x devices.

The LaunchReturn extension is a plug-in and has no properties.You must also install the LaunchReturnHelper.prc file on the device in addition to the SFE_LaunchReturn.prc file.

Platform(s): Palm OS version 5.0+ only

Sample project(s): LaunchReturn

Table 11.60 LaunchReturn extension methods

Method	Description
About	Displays information about the extension in a dialog box.
PrepLaunchReturn	Prepare to return to our app after switching to other app.
UnprepLaunchReturn	Cancel preparation to return to our app after switching to other app.
LaunchApp	Launch app via specified CreatorID.
PrepIncomingCall	Prepare to listen for incoming call notifications.
UnprepIncomingCall	Cancel preparation to listen for incoming call notifications.
CheckIncomingCall	Check to see if there is an incoming phone call.
IsTreoPhone	Check if device is a Treo Palm OS smartphone or not.

LaunchURL extension

The LaunchURL extension enables you to launch a specified URL in a web browser (Blazer for the Palm OS 5.x platform or Internet Explorer for the Pocket PC platform). It also enables you to view local html and image files in the web browser. The LaunchURL extension is a plug-in and has no properties. On the Pocket PC platform, the LaunchURL extension can also be used to open documents and media files in their default viewers, for example a Doc file in Pocket Word, or an MP3 audio file or WMV video file. To open documents and media files, pass the full path and name of the file in the URL parameter, instead of a standard URL.

Platform(s): Palm OS and Pocket PC

Sample project(s): LaunchURL

Table 11.61 LaunchURL extension methods

Method	Description
About	Displays information about the extension in a dialog box.
LaunchURL	Launch a specified URL or view local html and image files in the default device web browser.

Math extension

The Math extension provides mathematical capabilities to your Satellite Forms applications. The Math extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Advanced Calculator

Table 11.62 Math extension methods

Method	Description
About	Displays information about the extension in a dialog box.
ABS	Returns the absolute value of the specified number.
ACOS	Calculates the arc cosine of the specified number.
ACOSH	Calculates the hyperbolic arc cosine of the specified number.
ASIN	Calculates the arc sine of the specified number.
ASINH	Calculates the hyperbolic arc sine of the specified number.
ATAN	Calculates the arc tangent of the specified number.
ATAN2	Calculates the arc tangent of *x*/*y*.
ATANH	Calculates the hyperbolic arc tangent of the specified number.
CBRT	Calculates the cube root of the specified number.
COPYSIGN	Returns *x* with the sign of *y*.
COS	Calculates the cosine of the specified number.
COSH	Calculates the hyperbolic cosine of the specified number.
DREM	Calculates the remainder of *x*/*y*.
EXP	Calculates the exponential *e* to the *x*.
EXPM1	Calculates the exponential *e* to the *x* -1.
FREXPFRAC	Breaks given value into normalized fraction and an integral power of 2. Returns the fractional value.
FREXPINT	Breaks given value into normalized fraction and an integral power of 2. Returns the integer value.
ILOGB	Binary exponent of non-zero *x*. Returns an integer.
ISINF	Evaluates a number for its relationship to positive or negative infinity.

Table 11.62 Math extension methods *(Continued)*

LDEXP	Calculates exponential (*x* * 2) to the *y*.
LOG	Calculates the natural logarithm of *x*.
LOG10	Calculates the base ten logarithm of *x*.
LOG2	Calculates the base two logarithm of *x*.
MODFRAC	Breaks a floating-point number into integer and fractional parts, returning the fractional part.
MODFINT	Breaks a floating-point number into integer and fractional parts, returning the integer part.
POW	Calculates the exponential *x* to the *y*.
ROUND	Rounds a number to nearest decimal place specified.
SIN	Computes the sine of the specified number.
SINH	Computes the hyperbolic sine of the specified number.
SQROOT	Calculates the positive square root of the specified number.
TAN	Calculates the tangent of the specified number.
TANH	Calculates the hyperbolic tangent of the specified number.

Memory extension

The Memory extension provides functions to allocate, read, and write blocks of memory for your Satellite Forms applications. The Memory extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Memory

Table 11.63 Memory extension methods

Method	**Description**
MemoryAbout	Displays information about the Memory extension in a dialog box.
MemoryAllocate	Allocates dynamic memory.
MemoryCompare	Compares two blocks of memory.
MemoryCopy	Copies one block of dynamic memory to another.
MemoryFree	Frees memory allocated using `MemoryAllocate`.
MemoryGetByte	Returns the value of the specified byte in memory.
MemoryGetString	Returns the value of the specified string in memory.
MemoryReallocate	Resizes a block of dynamic memory.
MemoryReverse	Reverses a block of dynamic memory.
MemorySearch	Searches for a block of memory in another block of memory.
MemorySet	Sets a range of memory to the specified value.
MemorySetByte	Sets the value of the specified byte in memory.

Table 11.63 Memory extension methods *(Continued)*

MemorySetString	Sets the value of the specified string in memory.

Printer extension

The Printer extension provides printing capabilities for a Seiko DPU-414, Epson-compatible printer to your Satellite Forms applications. The Printer extension is a plug-in and has no properties.

Platform(s): Palm OS only

Sample project(s): Printer

Table 11.64 Printer extension methods

Method	Description
About	Displays information about the extension in a dialog box.
ClosePort	Closes the serial port, which saves battery power.
LBackSp	Sends a Backspace character to the printer.
LBell	Sends a Bell character to the printer.
LCancel	Sends a CAN character to the printer.
LCondensed	Sets the printer’s condensed print mode.
LDDGraphics	Sets double density graphics mode for the specified number of following bytes.
LDoubleStrike	Sets printer’s double strike print mode.
LDoubleWidth	Sets double width mode for `noChars` characters.
LDoubleWidthN	Sets the printer’s double width print mode.
LEmphasized	Sets the printer’s emphasized print mode.
LFormFeed	Sends a Form Feed character to printer.
LIntnlChars	Sets the printer’s international character set.
LLineFeed	Moves the print head the specified number of dots.
LLtMargin	Sets the left margin to the specified number of characters.
LPageLength	Set printer's page length to the specified number of lines.
LPrint	Prints a string.
LPrintCR	Prints a carriage return. Use `LPrintLn` to print the characters specified by `LSetAutoLF`.
LPrintDir	Sets the print direction.
LPrintF	Prints the specified text in a column of the specified width.
LPrintGraph	Prints graphics in single or double density mode.
LPrintLF	Prints a Line Feed character.
LPrintLN	Prints text followed by a Carriage Return and a Line Feed if `AutoLF` is set to TRUE.

Table 11.64 Printer extension methods *(Continued)*

LRepeatStr	Prints the specified string the specified number of times.
LReset	Sends a Reset command to the printer.
LRtMargin	Sets the right margin to the specified number of characters.
LSDGraphics	Sets single density graphics mode for the specified number of following bytes.
LSelectFont	Sets the font.
LSetAutoLF	Sets the `AutoLF` attribute.
LSetGraphics	Sets the printer's graphics mode and number of bytes. Supports quad density graphics.
LSetLine	Sets line height, usually 11 or 15 dots.
LSubscript	Sets the printer's subscript mode to the specified number of dots.
LSuperscript	Sets the printer's superscript mode to the specified number of dots.
LTab	Sends a Tab character to the printer.
OpenPort	Opens the serial port using the settings specified with `SetPort`. Opening the serial port uses more battery power.
SetPort	Sets the serial port parameters.
SetPrinter	No additional printers are supported. Must be set to 0.

Puma Beam DB extension

The Puma Beam DB extension provides infrared (IR) data file transmission capabilities to your Satellite Forms applications. The Puma Beam DB extension is a plug-in and has no properties.

Platform(s): Palm OS only

Sample project(s): Beam It!

Table 11.65 Puma Beam DB extension methods

Method	**Description**
PBD_About	Displays information about the extension in a dialog box.
PBD_BeamDb	Sends a Palm database file to another device using the IR port.
PBD_SendDbByName	Sends Palm Db specified by name, giving user option to select transport method (Beam, Bluetooth, SMS, VersaMail, etc.)
PBD_Version	Returns the version number of this extension.

Puma Data Manager extension

The Puma Data Manager extension provides access to a selected set of Palm OS Data Manager APIs to your Satellite Forms applications. The Puma Data Manager extension is a plug-in and has no properties.

Platform(s): Palm OS only

Sample project(s): View It!

Table 11.66 Puma Data Manager extension methods

Method	Description
PDM_About	Displays information about the extension in a dialog box.
PDM_DeleteDb	Deletes the specified database.
PDM_GetDbAppInfoID	Returns the app info ID of the `LastDb` cache (Local ID).
PDM_GetDbAttributes	Returns attributes of the `LastDb` cache.
PDM_GetDbBckUpDate	Returns the last back up date of the `LastDb` cache.
PDM_GetDbBckUpDateStr	Returns the last back up date of the `LastDb` cache as a string.
PDM_GetDbCardNo	Returns the card number of the `LastDb` cache.
PDM_GetDbCrDate	Returns the creation date of the `LastDb` cache.
PDM_GetDbCrDateStr	Returns the creation date of the `LastDb` cache as a string.
PDM_GetDbCreatorID	Returns the creator ID of the `LastDb` cache.
PDM_GetDbDataBytes	Returns the number of bytes in the data portion of the `LastDb` cache.
PDM_GetDbModDate	Returns the modification date of the `LastDb` cache.
PDM_GetDbModDateStr	Returns the modification date of the `LastDb` cache as a string.
PDM_GetDbModNum	Returns the number of modifications to the `LastDb` cache.
PDM_GetDbName	Returns the name of the last database loaded.
PDM_GetDbNumRecords	Returns the number of records contained in the `LastDb` cache.
PDM_GetDbSortInfoID	Returns the sort info ID of the `LastDb` cache.
PDM_GetDbTotalBytes	Returns the number of bytes in the `LastDb` cache.
PDM_GetDbType	Returns the type of the `LastDb` cache.
PDM_GetDbVersion	Returns the version number of the `LastDb` cache.
PDM_GetLastError	Returns the last error code.
PDM_GetNextDb	Returns local ID of the next database that matches the search criteria. Consecutive calls to this method traverses all databases matching the search criteria.
PDM_GetNumberOfMatchingDb	Returns the number of databases that match the search criteria.
PDM_LoadDb	Loads the specified database into the `LastDb` cache.
PDM_NewDbIterator	Starts a new search.
PDM_SetDbAttributes	Sets the attributes of the specified database.
PDM_Version	Returns the version number of this extension.

Puma Error Manager extension

The Puma Error Manager extension provides useful Palm error handling capabilities to your Satellite Forms applications. The Puma Error Manager extension is a plug-in and has no properties.

Platform(s): Palm OS only

Table 11.67 Puma Error Manager extension methods

Method	Description
PEM_About	Displays information about the extension in a dialog box.
PEM_GetErrorString	Converts an error code to a string.
PEM_Version	Returns the version number of this extension.

Puma Resource Manager extension

The Puma Resource Manager extension provides full access to Palm OS resources and their databases to your Satellite Forms applications. The Puma Resource Manager extension is a plug-in and has no properties.

Platform(s): Palm OS only

Sample project(s): Resource Viewer

Table 11.68 Puma Resource Manager extension methods

Method	Description
About	Displays information about the extension in a dialog box.
RM_AttachCurrentResource	Attaches the current resource to the database.
RM_CloseDatabase	Closes the current database.
RM_DetachedResToCurRes	Moves the detached resource to the current resource.
RM_DetachResource	Detaches the resource and stores the current pointer.
RM_FindResource	Finds the current resource by resource type and ID.
RM_FindResourceByIndex	Searches for the resource by type and index number.
RM_Get1Resource	Returns the resource from the most currently opened database.
RM_GetLastErrorNumber	Returns the last Resource Manager error code.
RM_GetResource	Returns the resource specified by the type and ID.
RM_GetResourceByIndex	Returns the resource specified by index.
RM_LockRes	Locks the current resource in memory.
RM_NewResource	Adds a new resource to the open database.
RM_NumResource	Returns the number of resources in the current database.
RM_OpenDatabase	Open the specified resource database.
RM_OpenDBNoOverlay	Open the specified resource database with no modifications to the overlay.

Table 11.68 Puma Resource Manager extension methods *(Continued)*

RM_PassResPtr	Returns a pointer to the memory block of a locked resource.
RM_ReleaseResource	Releases the current resource.
RM_RemoveResource	Removes the specified resource from the current database.
RM_ResizeResource	Resizes the current resource.
RM_ResourceInfo	Returns information about the specified resource.
RM_ResTypeInfo	Returns the resource type.
RM_SearchResource	Searches for a resource by type and local ID.
RM_SetResourceInfo	Sets the resource info for the current resource.
RM_UnlockRes	Unlocks the current resource. Use only after using `RM_LockRes`.

Px Screen Tool extension

The Px Screen Tool extension provides useful routines to manipulate the Palm display screen to your Satellite Forms applications. The Px Screen Tool extension is a plug-in and has no properties.

Platform(s): Palm OS only

Table 11.69 Px Screen Tool extension methods

Method	Description
About	Displays information about the extension in a dialog box.
RestoreScreenDepth	Restores Palm screen depth to the value before using `SetMaxScreenDepth`.
SetMaxScreenDepth	Sets the Palm screen to the maximum screen depth. Use `RestoreScreenDepth` to restore to the screen to its original depth. You can use this method to enable grayscale support on a monochrome Palm handheld.
Version	Returns the version number of this extension.

Random Number Generator extension

The Random Number Generator extension provides pseudo-random number generation capabilities to your Satellite Forms applications. The Random Number Generator extension has no properties. This extension is based on the `drand48` standard. This standard generates numbers through 48-bit arithmetic provided by the following equation: $X[I + 1]=(a*X[I] + c)mod(2^{48})$ where *a* equals `0x5DEECE66D` and *c* equals `0xB`. The Random Number Generator extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Random

Table 11.70 Random Number Generator extension methods

Method	Description

Table 11.70 Random Number Generator extension methods *(Continued)*

About	Displays information about the extension in a dialog box.
DRand48	Returns a pseudo-random number greater than 0 but less than or equal to 1.
Seed_Val	Returns the seed value entered for use with `SRand48`.
Seed_Val16V	Returns 16 bits of the three orders of the 48-bit seed value used with `Seed48`.
Seed48	Sets the 48-bit seed value.
SRand48	Assigns the highest 32 bits of X[0] to the specified integer value. Assigns the remaining 16 bits `0x330E`.
X_Buffer	Returns the current `X[i]` value from `X[i] = (a*X[i-1] + c)mod` 2^{48}.
X_Initial	Returns 16 bits of the initial 48-bit `X[i]` value.
X_Last	Returns the last `X[i-1]` value from `X[i] = (a*X[i-1] + c)mod` 2^{48}.

ScreenSize extension

The ScreenSize extension gives you information about the current screen size on the Pocket PC device. It can be used as a plug-in extension just by calling the GetScreenSize, GetScreenWidth, or GetScreenHeight script functions as desired, or it can be used as an SFX control that will fire an event when the screen size changes. There are no SFX control properties for the ScreenSize control.

Platform(s): Pocket PC only

Sample project(s): ScreenSize

Table 11.71 ScreenSize extension methods

Method	**Description**
About	Displays information about the extension in a dialog box.
GetScreenHeight	Return screen height.
GetScreenSize	Return screen dimensions in string WWWxHHH.
GetScreenWidth	Return screen width.

Serial Port extension

The Serial Port extension provides serial port read/write functionality to your Satellite Forms applications. The Serial Port extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Terminal

Table 11.72 Serial Port extension methods

Method	**Description**
SerialPortAbout	Displays information about the extension in a dialog box.

Table 11.72 Serial Port extension methods *(Continued)*

SerialPortAllocate	Creates an instance of a serial port.
SerialPortClose	Closes the specified serial port.
SerialPortConfigure	Configures the specified serial port.
SerialPortFree	Deletes an instance of a serial port.
SerialPortGetBytesAvailable	Returns the number of bytes available.
SerialPortGetLastError	Returns a string containing the last error message.
SerialPortOpenBinary	Opens the specified serial port in binary mode.
SerialPortOpenText	Opens the specified serial port in text mode.
SerialPortRead	Reads data from the specified serial port.
SerialPortReadByte	Reads one byte of data from the specified serial port.
SerialPortReadString	Reads a string of data from the specified serial port.
SerialPortSetBuffer	Sets the data buffer. The default data buffer is 512 bytes.
SerialPortWrite	Writes data to the specified serial port.
SerialPortWriteByte	Writes a byte of data to the specified serial port.
SerialPortWriteString	Writes a string of data to the specified serial port.

ShowImage control

The ShowImage extension is a custom control that provides Satellite Forms Pocket PC applications the ability to display common image files including JPG/GIF/PNG/BMP and more on the current form. Images are stretched/shrunk to fit the specified control rectangle. The ShowImage control can also act like a button control by handling pen taps if desired.

Platform(s): Pocket PC only

Sample project(s): ShowImage

Table 11.73 ShowImage control properties

Property	Description
VISIBLE	Determines whether the ShowImage control is visible (true) or hidden (false) on the form when the form opens.
BORDER	Determines whether a border is drawn around the ShowImage control.
DOBUTTONBEHAVIOR	Determines whether the ShowImage control acts like a button control by responding to pen taps.
IMAGEFILE	The path and name of the image file to display.

Table 11.74 ShowImage extension methods

Method	Description
About	Displays information about the extension in a dialog box.

Table 11.74 ShowImage extension methods *(Continued)*

GetBorder	Get BORDER property of ShowImage control.
GetButtonBehavior	Get DOBUTTONBEHAVIOR property of ShowImage control.
GetCtrlPosition	Get position and size of ShowImage control.
GetImageFile	Get IMAGEFILE property of ShowImage control.
GetVisible	Get VISIBLE property of ShowImage control.
SetBorder	Set BORDER property of ShowImage control.
SetButtonBehavior	Set DOBUTTONBEHAVIOR property of ShowImage control.
SetCtrlPosition	Set position and size of ShowImage control.
SetImageFile	Set IMAGEFILE property of ShowImage control.
SetVisible	Set VISIBLE property of ShowImage control (show or hide control).
ShowImageFile	Show an image file at the specified position by painting it directly onto the form (not as a custom control).

Slider control

The Slider control provides an additional control for use in Satellite Forms applications.

Platform(s): Palm OS and Pocket PC

Sample project(s): SFX Control

Note The Color Slider control is usually a better choice, as it supports color.

Table 11.75 Slider control properties

Property	Description
BEEP	Specifies whether the control should beep when the user interacts with it. If set 0, no beep occurs. If set to 1, a beep sounds when the user first touches the control. The default setting is 0.
DSOURCE	Fields("col_name") specifies that "col_name" (double-quotes required) in the form’s linked table that contains the control will be used to load and store the value of the control. The default value is “none”.
FASTTRACK	Specifies how often the `OnClick` event fires: • 0 = Every time the pen moves • 1 = After the pen is lifted The default setting is 0.
INITVAL	Specifies that the initial position of the control's bar should be at a number between MIN and MAX. The default setting is 0.
MIN	Specifies the minimum position of the control. The default setting is 0.
MAX	Specifies the maximum position of the control. The default setting is 100.

Table 11.75 Slider control properties *(Continued)*

STYLE	Specifies the style of slider control: • 0 = Progress Bar: no user interaction) • 1 = Slider with marker: user can slide the marker • 2 = Slider with no marker: user taps the bar to indicate the desired position • 3 = 3D Slider: color handhelds only The default setting is 1.
VISIBLE	Specifies whether the control is visible. The default value is TRUE.

Table 11.76 Slider control methods

Method	**Description**
About	Displays information about the control in a dialog box.
FastTrack	Specifies how often the `OnClick` event fires.
SldGetPosition	Returns the current position of the control.
SldSetMinMax	Sets the minimum and maximum values.
SldSetPosition	Sets the position of the control.
SetVisible	Makes the control visible or invisible.

SocketScan control

The SocketScan control interfaces to a Socket Mobile bar code reader card or Cordless Hand Scanner (CHS) on Palm OS and Pocket PC devices.

Platform(s): Palm OS and Pocket PC

Sample project(s): SocketScan

Table 11.77 SocketScan control properties

Property	**Description**
AFTERSCAN	Specifies a button control to execute immediately after performing the barcode scan, instead of executing the OnClick action of the SocketScan control.
BEFORESCAN	Specifies a button control to execute immediately before performing the barcode scan.
CHSINDICATOR	Specifies the indicator method to use on the CHS for a good scan.
CHSKEEPALIVE	Override the system auto off time *while* the PDA is connected to the CHS scanner.
SCANTRIGGER1 ... SCANTRIGGER4	Specifies a key code to trigger the barcode scan.

Table 11.78 SocketScan control methods

Method	**Description**

Table 11.78 SocketScan control methods *(Continued)*

About	Displays information about the control in a dialog box.
CHSConnect	Attempt to connect to the CHS via BlueTooth wireless.
CHSDisconnect	Disconnect from the CHS and close the BlueTooth connection.
CHSDoScan	Performs a scan with CHS.
CHSGetIndicator	Gets the current CHSINDICATOR value.
CHSGetKeepAlive	Gets the current CHSKEEPALIVE value.
CHSReady	Returns true or false to indicate if the CHS is connected and ready.
CHSSetIndicator	Sets the CHSINDICATOR value.
CHSSetKeepAlive	Sets the CHSKEEPALIVE value.
DoScan	Performs a scan and returns true or false to indicate if the scan was successful.
GetScanData	Returns the scanned data from the last scan operation.
GetScanOkay	Returns true or false to indicate if the last scan operation was successful.
GetScanParam	Returns the specified scanner parameter value.
GetScanTrigger	Returns a specified scan trigger keycode.
GetScanType	Returns the symbol code from the last scan operation.
ScannerDisable	Disnables the scanner. (Pocket PC only)
ScannerEnable	Enables the scanner. (Pocket PC only)
ScannerReady	Returns true or false to indicate if the scanner card is inserted and ready.
SetAfterScan	Specifies a button control or the SocketScan control's OnClick event to execute immediately after the scan event is fired.
SetBeforeScan	Specifies a button control to execute immediately before the scan event is fired.
SetScanParam	Sets the specified scanner parameter value.
SetScanToDefaults	Resets all the scanner parameters to factory defaults.
SetScanTrigger	Specifies a keycode that will trigger a scan.

Square Root extension

The Square Root extension provides a square root function to your Satellite Forms applications and serves as an example of how to write and implement an extension. The Square Root extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Square Root

Table 11.79 Square Root extension methods

Method	Description
About	Displays information about the extension in a dialog box.

Table 11.79 Square Root extension methods *(Continued)*

SqrRoot	Computes the square root of a number.

Strings extension

The Strings extension provides string manipulation functions for your Satellite Forms applications, similar to the string functions in Visual Basic. The Strings extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): Strings

Table 11.80 Strings extension methods

Method	Description
About	Displays information about the extension in a dialog box.
FormatDate	Formats a date according to the format expression.
FormatDateN	Formats a date value according to the format expression.
FormatTime	Formats a time according to the format expression.
FormatTimeN	Formats a time value according to the format expression.
HexStringFromInt	Converts an integer into a string of hex.
InStr	Finds occurrence of one string within another.
IntFromHexString	Converts a string of hex into an integer.
LCase	Converts string to all lower case.
LTrim	Trims leading spaces.
Pad	Pads a string with defined character to defined length.
RTrim	Trims trailing spaces.
Replace	Replace a substring with another substring, using a case sensitive comparison.
StrCompare	Performs case sensitive compare of two strings.
StrCompSort	Performs case sensitive compare of two strings.
String	Returns a repeating character string of the length specified.
SystemDateFormat	Returns system date format string.
SystemTimeFormat	Returns system time format string.
Trim	Trims leading and trailing spaces.
UCase	Converts string to all upper case.

Symbol Integrated Scanner control

The Symbol Integrated Scanner control interfaces to an integrated bar code reader on many Symbol and Janam Palm OS scanners, and Symbol and Janam Pocket PC handheld scanners.

Platform(s): Palm OS and Pocket PC

Sample project(s): Simple Scanner Demo, Bar Code, and Function Test

Table 11.81 Symbol Integrated Scanner control properties

Property	**Description**
AFTERSCAN	A Button control to execute after scan is received. This property is available only when you use the `Process` method.
BATTLOW	A control to execute after a `scanBatteryError` event. If no control is assigned, the default message appears. If the control's script includes the `FAIL` keyword, the default message does not appear.
BEEPAFTER	Beep after `GetScan` returns data. Valid values are "On" and "Off". This property is available only when you use the `Process` method.
BEFORESCAN	A Button to execute before scan input is saved but after `CtrlAdv` and `RecordAdv` operations are performed. Use to save the old value before it is modified or use to Cancel the current scan. This property is available only when you use the `Process` method.
CTRLADV	Automatically advance to next Edit control or record in sequence. Calls `RecordAdv` method when returning to the first Edit control. Valid values are "On" and "Off". This property is available only when you use the `Process` method.
RECORDADV	Sets the Record Advance mode.Valid values are: • "Off" Never changes the record. You must do so manually using the `AFTERSCAN` property. • "On" Advances to last record and stops. • "AlwaysCrt" Creates a new record after last Edit control. • "CrtAtEnd" Advance through the records, create new after last record.
SETFOCUS	Sets focus to the Bar Code Reader control after scan. Valid values are "On" and "Off". This property is available only when you use the `Process` method.

Table 11.82 Symbol Integrated Scanner control methods

Method	**Description**
About	Displays information about the control in a dialog box.
Aim	Sets the laser mode, either Aim or Scan.
AimDuration	
BarcodeLengths	Sets the scanner parameters.
BeepAfter	Beeps when the scan is complete.
BidirectionalRedundancy	Enables and disables bidirectional redundancy.
CheckDigit	Sets the Check Digit setting.
ClsiEditing	
Code32Prefix	

Table 11.82 Symbol Integrated Scanner control methods *(Continued)*

Code39CheckDigitVer	
Code39FullAscii	
CtrlAdv	Turns the `CTRLADV` property on or off.
DecodeLedOnTime	Sets the Decode LED on time property.
DecoderVersion	Returns the Symbol Decoder version as a string.
DecodeUpcEanRedun	
DecodeUpcEanSupp	
DisableScanner	Turns off the serial port and disables scanning. Uses less battery power.
EanZeroExtend	
EnableScanner	Turns on the serial port and enables scanning. Uses more battery power.
FollowCursor	Tapping a control makes it next to scan.
GetIndex	Returns the index of the current control.
GetRecordAdvMode	Returns the Record Advance Mode setting.
GetScan	Returns a string of scanned data.
GetType	Returns a string indicating the type of barcode just scanned.
GetTypeSettings	Returns the barcode types currently enabled.
HotSerResponseTimeout	
I2of5CheckDigit	
IsSymbolUnit	Determines whether the device is a Symbol unit.
LaserOnTime	
LedOn	Turns the LED on or off.
LinCodeTypeSecurLevel	
MsiPlesseyCkDigitAlg	
MsiPlesseyCkDigits	
NotisEditing	
PortDriverVersion	Returns a string indicating the Symbol Port Driver Version.
PrefixSuffixValues	
Process	Call from a Timer event to process automatic events.
RecordAdv	Sets the Record Advance mode.
ResetCtrls	Sets the index to the first control and cancels Record Advance mode.
RestoreDefaults	Restores scanner default settings.
ScanAngle	Selects Narrow or Wide scan angle.
ScanAvail	Returns the integer byte count in the receive buffer.

Table 11.82 Symbol Integrated Scanner control methods *(Continued)*

ScanManagerVersion	Returns a string indicating the Symbol Scan Manager version.
SetbarBOOKLAND_EAN	Enables or disables reading this barcode type.
SetbarCODABAR	Enables or disables reading this barcode type.
SetbarCODE128	Enables or disables reading this barcode type.
SetbarCODE39	Enables or disables reading this barcode type.
SetbarCODE93	Enables or disables reading this barcode type.
SetbarCOUPON	Enables or disables reading this barcode type.
SetbarD25	Enables or disables reading this barcode type.
SetbarEAN13	Enables or disables reading this barcode type.
SetbarEAN8	Enables or disables reading this barcode type.
SetBarExtra	Enables or disables extended barcode types not previously handled.
SetbarI2OF5	Enables or disables reading this barcode type.
SetbarISBT128	Enables or disables reading this barcode type.
SetbarMSI_PLESSEY	Enables or disables reading this barcode type.
SetbarPDF417	Enables or disables reading this barcode type.
SetbarTRIOPTIC	Enables or disables reading this barcode type.
SetbarUCC_EAN128	Enables or disables reading this barcode type.
SetbarUPCA	Enables or disables reading this barcode type.
SetbarUPCE	Enables or disables reading this barcode type.
SetbarUPCE1	Enables or disables reading this barcode type.
Set Convert	
SetFocus (Symbol Integrated Scanner)	Sets focus to the Bar Code Reader control after scan.
SetIndex	Sets the index of the current control. Uses the next control for the next scan.
SetPort	Ignored. Provided for compatibility only.
SetTermChar	Ignored. Provided for compatibility only.
SetTypeSettings	Sets the barcode types to be enabled.
SkipAdvance	Indicates whether to stay on same control for the next scan.
StartDecode	Starts scanning with the specified laser mode.
StripTerm	Ignored. Provided for compatibility only.
TermRecd	Returns TRUE if the termination character is last character in the string. Use after `GetScan` to verify that the termination character was received.
TransmissionFormat	
TransmitCodeIDChar	

Table 11.82 Symbol Integrated Scanner control methods *(Continued)*

TriggerMode	Sets the triggering mode.
UpcEanSecurLevel	
UpcPreamble	

Symbol MSR control

The Symbol MSR (Magnetic Stripe Reader) control interfaces a Symbol Palm OS scanner to a magnetic stripe reader through the serial port.

Platform(s): Palm OS only

Sample project(s): MSR

Table 11.83 Symbol MSR control methods

Method	Description
mAbout	Displays information about the control in a dialog box.
mArmToRead	Enables the MSR to be ready for a card swipe in buffered mode.
mClose	Closes the MSR Manager Library and frees resources.
mGetAddedField	Returns the MSR added field string for the specified field number.
mGetBufferMode	Returns the MSR buffer mode setting.
mGetDataBuffer	Requests card data using the current Buffered Mode setting.
mGetDataEditSetting	Returns the MSR data edit setting.
mGetDecoderMode	Returns the MSR decoder mode setting.
mGetFlexibleField	Returns the MSR flexible field string for the specified field number.
mGetLastError	Returns the error status of the last MSR function executed.
mGetLibVersion	Returns the MSR 3000 software library version.
mGetLRCSetting	Returns the MSR LRC setting.
mGetMSRVersion	Returns the MSR 3000 version.
mGetPostamble	Returns the MSR postamble string.
mGetPreamble	Returns the MSR preamble string.
mGetReservedChars	Returns the MSR reserved character information for the specified character number.
mGetSendCmd	Returns the MSR send command string for the specified command number.
mGetStatus	Returns the status of the last magnetic stripe read.
mGetTerminator	Returns the MSR terminator setting.
mGetTrackFormat	Returns the MSR track format string for the specified track number.
mGetTrackSelection	Returns the MSR track selection setting.
mGetTrackSeparator	Returns the MSR track separator character.

Table 11.83 Symbol MSR control methods *(Continued)*

mOpen	Loads and initializes the MSR 3000 Manager Library.
mReadMSRBuffer	Requests Card Information until receiving data or timeout occurs.
mReadMSRUnbuffer	Requests card data using unuffered mode.
mSelfDiagnose	Initiates the MSR 3000 self test and returns the results.
mSetAddedField	Sets added field strings for the MSR 3000.
mSetBufferMode	Sets buffered or unbuffered mode for the MSR 3000.
mSetDataEdit	Sets data edit mode.
mSetDataEditSend	Sends data edit send commands to the MSR 3000.
mSetDecoderMode	Sets the decoder mode for the MSR 3000.
mSetDefault	Resets the MSR 3000 to its default settings.
mSetFlexibleField	Sets flexible fields for data edit to the MSR 3000.
mSetLRC	Sets LRC mode for the MSR 3000.
mSetPostamble	Sets the postamble string for the MSR 3000.
mSetPreamble	Sets the preamble string for the MSR 3000.
mSetReservedChar	Sets the Special Reserved Characters for Generic Decoder only.
mSetTerminator	Sets the terminator character setting for the MSR 3000.
mSetTrackFormat	Sets the parameters for the Generic Decoder, including the Bit Format, Start and End Sentinel.
mSetTrackSelection	Sets the tracks that the MSR 3000 is to decode.
mSetTrackSeparator	Sets the track separator character for the MSR 3000.

SysUtils extension

The SysUtils extension provides access to OS (operating system) utility functions for Palm OS and Pocket PC platforms. The SysUtils extension is a plug-in and has no properties.

Platform(s): Palm OS and Pocket PC

Sample project(s): SysUtils

Table 11.84 SysUtils extension methods

Method	Description
About	Displays information about the extension in a dialog box.
SU_BlockAllHotKeys	Blocks all hotkey keypresses, but does not send a keypress that can be caught in the OnKey event.
SU_CheckSystemPassword	Returns 1 (true) or 0 (false) depending if the passed string is the current system password or not.
SU_ClipboardTextGet	Get the current text clip from the system clipboard.
SU_ClipboardTextSet	Paste the passed string into the system clipboard.

Table 11.84 SysUtils extension methods *(Continued)*

SU_DelAppPref	Deletes the saved preference given creatorID and pref index.
SU_GetAppPref	Returns the saved preference value given creatorID and pref index.
SU_GetBatteryPercent	Returns current battery charge level as a percentage (0-100) of full.
SU_GetDeviceID	Get the device unique ID string.
SU_GetDeviceModel	Get the device model string to help identify the device.
SU_GetMemInfo	Get the amount of available memory.
SU_GetOSVersion	Get the OS version string for the current device.
SU_GetOwnerName	Get the device owner name.
SU_GetPlatform	Get the device OS platform string, either PALMOS or POCKETPC.
SU_GetPluggedIn	Returns 1 (true) if AC power is currently connected or 0 (false) if not.
SU_HideStartIcon	(Pocket PC only) Hide the Pocket PC Start icon in the taskbar.
SU_HotSync	Initiates standard cradle hotsync by enqueuing HotSync virtual keypress.
SU_LaunchApp	Launch a specified application/document/URL, and pass an optional parameter.
SU_LaunchAppAtEvent	Launch a specified application/document/URL at a specified system event.
SU_LaunchAppAtTime	Launch a specified application/document/URL at a specified date and time.
SU_ModemHotSync	Initiates standard modem hotsync by enqueuing Modem HotSync virtual keypress.
SU_ParseDelimText	Returns a chunk of data in a string of delimited items.
SU_PasteChars	Paste a string to the keyboard input queue as though it was typed in. Input goes to the control that has the focus.
SU_PlaySoundFile	(Pocket PC only) Play a WAV audio file.
SU_PowerOff	Power off the device now, as if the power button had been pressed.
SU_QueueVirtualKey	Post a virtual key to the keyboard input queue. Input goes to the control that has the focus.
SU_RegDeleteKey	Delete specified key from registry and all settings within it.
SU_RegReadKey	Read specified key setting value from the registry.
SU_RegWriteKey	Write specified key setting value to the registry.
SU_Reset	Soft reset the device.
SU_SetAppPref	Saves the preference value for the supplied creatorID and pref index.
SU_SetAutoOffTime	Sets the auto-off timer on the Palm.
SU_SetDeviceDateTime	Set the device date and time to the passed value

Table 11.84 SysUtils extension methods *(Continued)*

SU_SetHotKey	Trap a hotkey keypress so that it sends a keypress that can be caught in the OnKey event, instead of the system getting it first.
SU_SysIdleTimerReset	Reset the system idle timer to prevent the device from dozing off into sleep mode.
SU_TapScreen	Enqueue a virtual pen tap at a screen coordinate.

TCPIP Winsock/Internet extension

The TCPIP Winsock extension for Pocket PC and Internet extension for Palm OS enable you to connect to TCP/IP based networks (such as the Internet). It is a minimal implementation of the Berkeley sockets standard, with which your application can connect to hosts and send/receive data. The TCPIP Winsock & Internet extensions are plug-ins and have no properties. The design and operation of these two extensions are very similar, so they are listed here together.

Platform(s): Palm OS and Pocket PC

Sample project(s): TCPIP Sockets, SMTP

Table 11.85 TCPIP Winsock/Internet extension methods

Method	Description
About	Displays information about the extension in a dialog box.
Close	Closes a socket.
CloseNetLib	Closes the net library. (PALM OS only)
Connect	Connects a socket to a specified address and port.
ConvertDottedToInetAddr	Converts a dotted decimal address (aaa.bbb.ccc.ddd) into a 32-bit network address." (PALM OS only)
ConvertInetAddrToDotted	Converts a 32-bit network address into a dotted decimal address (aaa.bbb.ccc.ddd). (PALM OS only)
GetDomainName	Retrieves the default domain added to names before DNS lookups occur. (PALM OS only)
GetHostByAddr	Performs a reverse DNS lookup on the passed IP address, returning the name of the host machine.
GetHostByName	Performs a DNS lookup on the passed host name.
GetLastError	Retrieves the last error encountered by an extension method.
GetRecvTimeout	Retrieves timeout value for Receive operations. (PALM OS only)
GetSendTimeout	Retrieves timeout value for Send operations. (PALM OS only)
GetServiceByName	Attempts to return the port number associated with a specified service. (PALM OS only)
GetSocketLinger	Gets the linger option for a socket.
OpenNetLib	Opens the net library. (PALM OS only)
OpenNetworkPrefPanel	Opens the standard Network Preference panel. (PALM OS only)
Receive	Receives data from a socket.

Table 11.85 TCPIP Winsock/Internet extension methods *(Continued)*

Send	Sends data to a socket.
SetDomainName	Sets the default domain name added to names before DNS lookups occur. (PALM OS only)
SetRecvTimeout	Sets timeout value for Receive operations. (PALM OS only)
SetSendTimeout	Sets timeout value for Send operations. (PALM OS only)
SetSocketLinger	Sets the linger option on a socket.
Shutdown	Shuts down a socket.
SKT_GetErrorString	Convert the specified error to a string. (PALM OS only)
SKT_SetAutoOffTime	Resets the automatic sleep timer. (PALM OS only)
Socket	Creates a new socket.
WSACleanup	De-initializes Winsock. (Pocket PC only)
WSAStartup	Initializes Winsock for use. (Pocket PC only)

UnitechScan control

The UnitechScan control interfaces to an integrated bar code scanner on many Unitech Pocket PC devices.

Platform(s): Pocket PC (Unitech devices only)

Sample project(s): UnitechScan

Table 11.86 UnitechScan control properties

Property	Description
AFTERSCAN	Specifies a button control to execute immediately after performing the barcode scan, instead of executing the OnClick action of the UnitechScan control.
GOODREADSOUND	Specifies whether you want to have the good read beep sound played on a successful scan.
LEDONTIME	Sets length of time in milliseconds to show green LED on a good scan.

Table 11.87 UnitechScan control methods

Method	Description
About	Displays information about the control in a dialog box.
GetGoodReadSound	Returns whether scanner is set to play the system good read sound when a good scan is received.
GetLEDOnTime	Returns length of time in milliseconds to show green LED when a good scan is received.
GetScanData	Returns the scanned data from the last scan operation.
GetScanDataLen	Returns the length in bytes of the scanned data from the last scan operation.

Table 11.87 UnitechScan control methods *(Continued)*

GetScanOkay	Returns true or false to indicate if the last scan operation was successful.
GetScanType	Returns the symbol code from the last scan operation.
IsScan2KeyEnabled	Returns whether Unitech Scan2Key utility is currently enabled.
IsUnitechScanner	Returns whether current device is a Unitech scanner.
ResetScanner	Resets the scanner.
Scan2KeyEnable	Enables or disables the Unitech Scan2Key utility.
ScannerDisable	Disables the scanner.
ScannerEnable	Enables the scanner.
SetAfterScan	Specifies a button control or the UnitechScan control's OnClick event to execute immediately after the scan event is fired.
SetGoodReadSound	Sets whether to play the system good read sound when a good scan is received.
SetLEDOnTime	Sets length of time in milliseconds to show green LED when a good scan is received.

WM5Camera extension

The WM5Camera extension is a plugin extension for Pocket PC that enables your Windows Mobile 5 and higher applications to capture photos or videos on camera-equipped devices. Photos are saved to .JPG files in the folder you specify. Videos should be saved to .3GP files in order to be compatible with MMS standards among mobile phones.

Platform(s): Pocket PC only (Window Mobile 5+ OS version)

Sample project(s): WM5Camera

Table 11.88 WM5Camera extension methods

Method	**Description**
About	Displays information about the extension in a dialog box.
CameraCapture	Capture photo or video from camera, return result code.

Accessing properties

To access an object’s properties, use the following syntax:

```
object.[object.,...].Property
```

You can always specify the full object hierarchy. The following example sets the value of Variable to the data stored in the Value field of the Orders table for the current record:

```
Variable = Tables("Orders").Fields("Value").Data
```

You do not need to specify the default property. For the Fields object, Data is the default property. The following syntax is equivalent to the example above:

```
Variable = Tables("Orders").Fields("Value")
```

You also do not need to specify objects that are implied. The Table object linked to the current form is implied. If you are referencing this table, the following statement is equivalent to the example above:

```
Variable = Fields("Value")
```

Note You must always specify at least one object when accessing a property.

Using methods

To access an object’s methods, use the following syntax:

```
object.[object.,...].Method
```

You can always specify the full object hierarchy. The following example sets the value of Variable to the sum of the data stored in the Value column of the Orders table for all records in the Orders table:

```
Variable = Tables("Orders").Sum("Value")
```

You do not need to specify objects that are implied. The Table object linked to the current form is implied. If you are referencing this table, the following statement is equivalent to the example above:

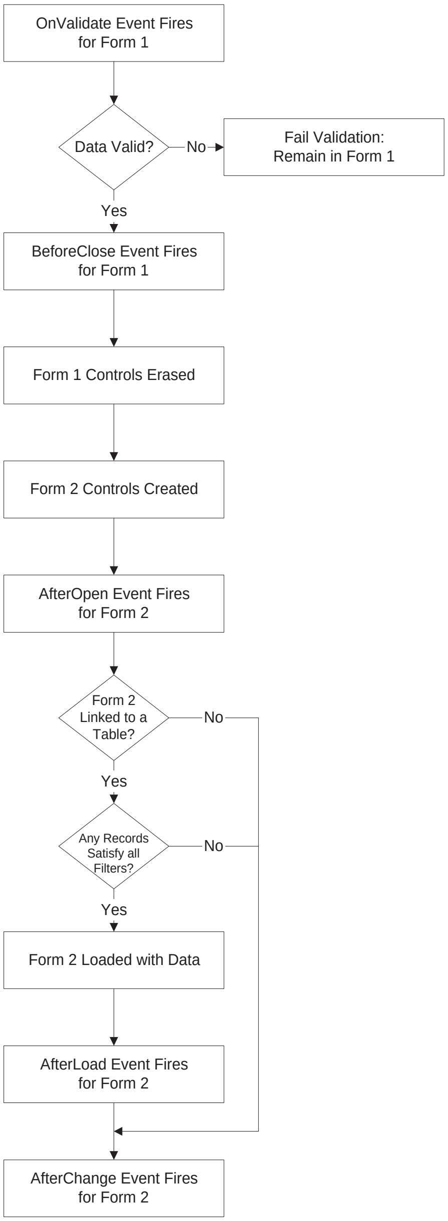
```
Variable = Tables().Sum("Value")
```

The only methods that do not require an object are the methods of the application: `FormatNumber`, `GetSysTime`, `MsgBox`, `Prompt`, etc.. For all other methods, you must specify the object.

Understanding events

The following figure illustrates the order in which Satellite Forms events occur for the general case of a jump from one form (Form1) to another form (Form2).

Figure 11.2 Example of Satellite Forms event flow


OnValidate Event Fires for Form 1
Data Valid?
No
Fail Validation: Remain in Form 1
Yes
BeforeClose Event Fires for Form 1
Form 1 Controls Erased
Form 2 Controls Created
AfterOpen Event Fires for Form 2
Form 2 Linked to a Table?
No
Yes
Any Records Satisfy all Filters?
No
Yes
Form 2 Loaded with Data
AfterLoad Event Fires for Form 2
AfterChange Event Fires for Form 2

Creating a Satellite Forms script

This section provides a brief example of how to use scripting in Satellite Forms. This section assumes you have a working knowledge of Microsoft Visual Basic®. If you need additional information on how to program with Visual Basic, any good bookstore that carries computer books typically has several books on programming with Microsoft Visual Basic. The Satellite Forms own scripting language is essentially a subset of Visual Basic and shares much of its syntax. See Satellite Forms scripting language reference on page 314 for detailed syntax and usage information.

To open the Script editor, select **View > Show Scripts** from the MobileApp Designer menu, click the **Show Scripts** button on the Misc Toolbar, or press **Ctrl + T** on your keyboard. If you have already set the action for a control to Run Script, you can right-click the control either on the desktop or in the Workspace palette and select **Show Control Scripts** from the context menu.

Satellite Forms scripting is event-driven. Click the Scripts tab in the Workspace palette to access all scripts currently available in the project. The scripts are organized under icons for each form and menu in the project, as well as an icon labelled Global that contains application-global scripts. All script names are based on the event that runs the script.

Each Form icon contains a series of event icons, each of which can execute a script. Any control on a form for which you have set the action to Run Script also has an event icon with a name that follows this pattern: OnClick <*ControlName*>.

Satellite Forms also supports both local and global variables. Variables are created when you use the `Dim` statement to declare them. All variables are of type `variant` when you first declare them. When you assign a value to a variable, the variable's type changes to match the assigned value, for example string, integer, or floating-point. You can convert a variable from one type to another by using the `Float`, `Int`, and `Str` functions. Variables you declare as in the Global icon tree are accessible to all forms in an application. Variables you declare within a script for a particular event are local in scope, which means you can only access them within that script, and only exist for the duration of the script.

To create a script, simply type your code in the Script editing window for the form and event you desire.

Procedure Create your first script

1 Open a new project.

2 Add an edit control, **InputA**, to Form 1 of the project.

3 Add a button, **Button1**, to Form 1 of the application.

4 Set the action for Button1 to **Run Script**.

5 Click the **Edit Script...** button on the Control Actions and Filters dialog box.

6 Type the following code in the scripting window:

```
'Example of Scripting in Satellite Forms
If InputA < 10 or InputA > 100 then
      MsgBox("You must enter a number between 10 and 100.")
EndIf
```

7 Select **Build > Compile Script**, click the **Compile Script** button on the Misc toolbar, or press **Ctrl + F7** to compile the script.

8 Download the application to the handheld and run the application. MobileApp Designer prompts you to save and then HotSync the application.

9 Test the script by running the application, entering data into **InputA**, and clicking **Button1**.

Satellite Forms scripting language keywords and operators

The Satellite Forms scripting language provides many keywords and operators in addition to the object properties and methods. The following tables summarize these language elements by category. The next section provides an alphabetical listing and description of all keywords, operators, object properties, and methods.

Satellite Forms scripting language keywords

The following table lists and describes the Satellite Forms scripting language keywords:

Table 11.89 Satellite Forms scripting language keywords

Keyword	Description
Dim	Dimensions (declares) a variable.
Exit	Exits a script with success.
Fail	Exits a script with failure.
For…To…Next	Performs a For loop.
Function	Defines a Global script function.
If … Then…Else… ElseIf…EndIf	Conditionally executes statements.
Quit	Exit the application and return to the app launcher.
Sub	Defines a Global script subroutine.
While … Wend	Performs a While loop.

Satellite Forms scripting language conversion operators

The following table lists and describes the Satellite Forms scripting language conversion operators:

Table 11.90 Satellite Forms scripting language conversion operators

Conversion operator	Description
Bool	Converts a variable to a Boolean.
Empty	Assigns a variable or table field to Empty state.
Float	Converts a variable to floating point.
Int	Converts a variable to an integer.
Int64	Converts a value to a 64-bit integer.
Str	Converts a variable to a string.

Satellite Forms scripting language comparison operators

The following table lists and describes the Satellite Forms scripting language comparison operators:

Table 11.91 Satellite Forms scripting language comparison operators

Comparison operator	Description
= [equal]	Assigns a value to a variable or evaluates equality of two values.
> [greater than]	Evaluates whether a value is greater than another value.
>= [greater than or equal]	Evaluates whether a value is greater than or equal to another value.
< [less than]	Evaluates whether a value is less than another value.
<= [less than or equal]	Evaluates whether a value is less than or equal to another value.
<> [not equal]	Evaluates whether two values are not equal.
IsEmpty	Evaluates whether a variable or table field is Empty.

Satellite Forms scripting language arithmetic operators

The following table lists and describes the Satellite Forms scripting language arithmetic operators:

Table 11.92 Satellite Forms scripting language arithmetic operators

Arithmetic operator	Description
+ [add]	Adds two numbers.
- [subtract]	Subtracts one number from another.
* [multiply]	Multiplies two numbers.
/ [divide, float]	Divides one number by another; returns a floating-point value.
\ [divide, integer]	Divides one number by another; returns an integer value.
Mod	Returns remainder from integer division.

Note Floating point numbers are limited to a total of 15 digits.

Satellite Forms scripting language logical and bitwise operators

The following table lists and describes the Satellite Forms scripting language logical and bitwise operators:

Table 11.93 Satellite Forms scripting language logical and bitwise operators

Logical or bitwise operator	Description
Not [logical] Not [bitwise]	When used with Booleans or conditions, negates a condition, value, or variable. When used with a numerical argument, performs a bitwise Not on the argument.
And [bitwise]And [logical] And [bitwise]And [logical]	When used with Booleans or conditions, joins two conditions where both conditions must evaluate to TRUE for the expression to evaluate to TRUE. When used with numerical arguments, performs a bitwise And between the arguments.
Or [logical] Or [bitwise]	When used with Booleans or conditions, joins two conditions where only one condition must evaluate to TRUE for the expression to evaluate to TRUE. When used with numerical arguments, performs a bitwise Or between the arguments.
Xor [logical] Xor [bitwise]	When used with Booleans or conditions, joins two conditions where one and only one condition must evaluate to TRUE for the expression to evaluate to TRUE. When used with numerical arguments, performs a bitwise Xor between the arguments.

Satellite Forms scripting language string operators

The following table lists and describes the Satellite Forms scripting language string operators:

Table 11.94 Satellite Forms scripting language string operators

String operator	Description
& [concatenate]	Concatenates two strings.
Left	Returns the leftmost characters of a string.
Len	Returns the length of a string.
Mid operator	Returns a subset of characters of a string.
Mid statement	Replaces a subset of characters in a string.
Right	Returns the rightmost characters of a string.

Satellite Forms scripting language miscellaneous operators

The following table lists and describes miscellaneous Satellite Forms scripting language operators.

Table 11.95 Satellite Forms scripting language miscellaneous operators

Operator	Description
' [comment]	Marks a comment that is not compiled.
_ [continue line]	Enables a line of code to span multiple lines.
. [method, property]	Accesses an object's properties or executes its methods.
Asc	Returns the ASCII value of a character.
Chr	Returns the character associated with an ASCII value.

Satellite Forms scripting language reference

This section provides an alphabetical listing of all keywords, properties, methods, events, and operators the Satellite Forms scripting language supports. Each listing provides detailed information and many provide sample code.

You may want to try some of the sample code yourself. You can do this by creating a sample application containing the following objects:

- Twelve edit controls named **InputA** through **InputF** and **OutputA** through **OutputF**.
- Two buttons named **Button1** and **Button2** with their action properties set to **Run Script.**
- One table called **Emps**, with two columns called **Name** and **Salary**.

This application is sufficient to run the majority of the samples. Some samples may require you to add additional controls or link controls to columns or types of columns.

+ [add]

Operand1 + Operand2

Adds two values or variables.

Parameters *Operand1* Value or variable to be added to *Operand2.*

Operand2 Value or variable to be added to *Operand1.*

Return Value The sum of *Operand 1* and *Operand2.*

Comments If both operands are integers, the result is an integer.

If either operand is floating point, the result is floating point.

If either operand is a string, the string operand is first converted to either an integer or a floating point depending on the operand's value. For example, the string "2" is converted to an integer, while the string "2.4" is converted to a floating point number. The operation then proceeds and the result is the same as described above.

For floating-point calculations, the result is formatted to have the least number of decimal places necessary. For example, if you add 2.10 and 2.00, the result is 4.1. Use the `FormatNumber` method to format the result to the desired number of decimal places.

Example

```
'Example of Addition
'InputA, InputB, and OutputA 'are edit controls.
OutputA = InputA + InputB
```

See Also // [divide, float], \ [divide, integer], FormatNumber, Mod, * [multiply], - [subtract]

' [comment]

' *Comment*

An Apostrophe anywhere in a line makes the rest of the line a comment.

Parameter *Comment* Text of comment

Return Value None

Comments The compiler ignores Comments in scripts.

Example

```
'Example of a comment
'InputA is an edit control.
InputA = 1 'Comments can also start here.
```

& [concatenate]

Operand1 & Operand2

Concatenates two string values or variables together.

Parameters *Operand1* Value or variable to be concatenated with *Operand2.*

Operand2 Value or variable to be concatenated with *Operand1.*

Return Value String value resulting from concatenation of the two input values or variables.

Comments Both operands are converted to strings, if they are not strings already, before they are concatenated.

Example

```
'Example of Concatenation
'InputA, InputB, and OutputA are edit controls.
OutputA = InputA & InputB
MsgBox ("InputA contains " & InputA)
```

_ [continue line]

Line1_
Line2

The Underscore at the end of a line continues the line of code onto the next line.

Parameters *Line1* Line of code.

Line2 Continuation of *Line1.*

Return Value None

Example

```
'Example of code spanning multiple lines
'InputA, InputB, OutputA, and
'OutputB are edit controls.
'Add on single line.
OutputA = InputA + InputB
'Add over multiple lines.
OutputB = InputA + _
InputB
```

/ [divide, float]

Operand1 / Operand2

Divides one value or variable by another and returns a floating-point quotient.

Parameters *Operand1* Value or variable to be divided by *Operand2*.

Operand2 Value or variable by which *Operand1* is divided.

Return Value The floating-point quotient of the division of *Operand1* by *Operand2.*

Comments Both operands are converted to floating point; the result is a floating-point number.

The result is formatted to have the least number of decimal places necessary. For example, if you divide 4.20 by 2.00, the result is 2.1. Use the `FormatNumber` method to format the result to the desired number of decimal places.

Example

```
'Example of floating-point division
'InputA, InputB, and OutputA are edit controls.
OutputA = InputA / InputB
```

See Also + [add], \ [divide, integer], , Mod, * [multiply], - [subtract]

\ [divide, integer]

Operand1 \ Operand2

Divides one value or variable by another and returns an integer quotient.

Parameters *Operand1* Value or variable to be divided by *Operand2*.

Operand2 Value or variable by which *Operand1* is divided.

Return Value The integer quotient of the division of *Operand1* by *Operand2.*

Comments Both operands are converted to integers if they are not integers already. Decimal places are truncated. The result is an integer. Use the `Mod` operator to determine the remainder of integer division.

Example

```
'Example of integer division
'InputA, InputB, and OutputA are edit controls.
OutputA = InputA \ InputB
OutputB = InputA Mod InputB
```

See Also + [add], / [divide, float], , Mod, * [multiply], - [subtract]

= [equal]

Operand1* = *Operand2

Assigns a value to a variable or evaluates the equality of two values or variables.

Parameters *Operand1* Assignments: Variable to which the value *Operand2* is assigned. Evaluations: Value that is tested against *Operand2*.

Operand2 Assignments: Value that is assigned to *Operand1.* Evaluations: Value that is tested against *Operand1.*

Return Value None for assignments.
TRUE or FALSE for evaluations of equality.

Example

```
'Example of assignment and test of equality
'InputA, InputB, InputC, InputD, and OutputA
'are edit controls.
'Test of equality
If InputA  = InputB Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
'Assignment
InputC = 3
InputD = "ABC"
```

See Also > [greater than], >= [greater than or equal], , < [less than], <= [less than or equal], <> [not equal]

> [greater than]

Operand1* > *Operand2

Evaluates whether one value or variable is greater than another.

Parameters *Operand1* Value or variable to be evaluated against *Operand2.*

Operand2 Value or variable to be evaluated against *Operand1.*

Return Value TRUE if *Operand1* is greater than *Operand2*; FALSE if not.

Comments For numeric values, higher numbers are greater than lower numbers.

For string values, strings that are later alphabetically are greater than strings that are earlier alphabetically.

Example

```
'Example of Greater Than comparison
'InputA, InputB, and OutputA are edit controls.
If Float(InputA) > Float(InputB) Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also = [equal], >= [greater than or equal] , < [less than], <= [less than or equal], <> [not equal]

>= [greater than or equal]

Operand1 >= Operand2

Evaluates whether one value or variable is greater than or equal to another.

Parameters *Operand1* Value or variable to be evaluated against *Operand2.*

Operand2 Value or variable to be evaluated against *Operand1.*

Return Value TRUE if *Operand1* is greater than or equal to *Operand2*; FALSE if not.

Comments For numeric values, higher numbers are greater than lower numbers.

For string values, strings that are later alphabetically are greater than strings that are earlier alphabetically.

Example

```
'Example of Greater Than or Equal comparison
'InputA, InputB, and OutputA are edit controls.
If Float(InputA) >= Float(InputB) Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also = [equal], > [greater than], < [less than], <= [less than or equal], <> [not equal]

< [less than]

Operand1 < Operand2

Evaluates whether one value or variable is less than another.

Parameters *Operand1* Value or variable to be evaluated against *Operand2.*

Operand2 Value or variable to be evaluated against *Operand1.*

Return Value TRUE if *Operand1* is less than *Operand2*; FALSE if not.

Comments For numeric values, lower numbers are less than higher numbers.

For string values, strings that are earlier alphabetically are less than strings that are later alphabetically.

Example

```
'Example of Less Than comparison
'InputA, InputB, and OutputA are edit controls.
If Float(InputA) < Float(InputB) Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also]= [equal], > [greater than], >= [greater than or equal], <= [less than or equal], <> [not equal]

<= [less than or equal]

Operand1 <= Operand2

Evaluates whether one value or variable is less than or equal to another.

Parameters *Operand1* Value or variable to be evaluated against *Operand2.*

Operand2 Value or variable to be evaluated against *Operand1.*

Return Value TRUE if *Operand1* is less than or equal to *Operand2*; FALSE if not.

Comments For numeric values, lower numbers are less than higher numbers.

For string values, strings that are earlier alphabetically are less than strings that are later alphabetically.

Example

```
'Example of Less Than or Equal comparison
'InputA, InputB, and OutputA are edit controls.
If Float(InputA) <= Float(InputB) Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also = [equal], > [greater than], >= [greater than or equal], < [less than], <> [not equal]

. [method, property]

Object.Method
Object.Property

Accesses an object’s properties or executes an object’s methods.

Parameter *Object* An object.

Return Value None

Example

```
'Example of using methods and accessing properties
'InputA, OutputA, OutputB, and OutputC are edit controls.
'Emps is a table.
'Use the Sum method of the table object.
OutputA = Tables("Emps").Sum("Salary")
'Access the Data property of the control.
InputA.
'Assign the result to the Data property of
'the control OutputB. Note that the Data property
'is default and can be omitted.
OutputB.Data = InputA.Data
OutputC = InputA
```

See Also Beep, Count, CurrentPage. Data, Delay, ExecAction, FormatNumber, GetSysTime, Index. MoveCurrent, MoveFirst, MoveLast, MoveNext, MovePrevious. MsgBox, Prompt, RecordValid, Show. Sum, Tone, Visible

<> [not equal]

Operand1 <> Operand2

Evaluates whether one value or variable is not equal to another.

Parameters	*Operand1*	Value or variable to be evaluated against *Operand2.*
	Operand2	Value or variable to be evaluated against *Operand1.*

Return Value TRUE if *Operand1* and *Operand2* are not equal; FALSE if not.

Example

```
'Example of Not Equal comparison
'InputA, InputB, and OutputA are edit controls.
If InputA <> InputB Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also = [equal], > [greater than], >= [greater than or equal], < [less than], <= [less than or equal]

* [multiply]

Operand1 * Operand2

Multiplies two values or variables.

Parameters	*Operand1*	Value or variable to multiply by *Operand2*.
	Operand2	Value or variable to multiply by *Operand1.*

Return Value The product of the multiplication of *Operand1* and *Operand2.*

Comments If both operands are integers, the result is an integer.

If either operand is floating point, the result is floating point.

If either operand is a string, that operand is first converted to either an integer or a floating point depending on its value. For example, the string “2” is converted to an integer, while the string “2.4” is converted to floating point. The operation then proceeds and the result is the same as described above.

If you are multiplying very large integers, ensure that the result of the multiplication operation does not exceed the allowed range for integers (-2147483648 to +2147483647). Use the `Float` conversion operator to convert at least one of the operands to a floating-point number to ensure that the answer is a floating-point number. The range for both positive and negative floating-point numbers (2.2250738585072014E-308 to 1.7976931348623158E+308) is significantly larger than the range for integers.

For floating-point calculations, the result is formatted to have the least number of decimal places necessary. For example, if you multiply 2.00 by 2.1, the result is 4.2. You can use the `FormatNumber` method to format the result to the desired number of decimal places.

Example

```
'Example of Multiplication
'InputA, InputB, and OutputA are edit controls.
OutputA = InputA * InputB
```

See Also + [add], / [divide, float], \ [divide, integer], Float, Mod, - [subtract]

- [subtract]

Operand1 – Operand2

Subtracts one value or variable from another.

Parameters *Operand1* Value or variable from which *Operand2* is subtracted.

Operand2 Value or variable which is subtracted from *Operand1.*

Return Value Difference of the two operands.

Comments If both operands are integers, the result is an integer.

If either operand is floating point, the result is floating point.

If either operand is a string, that operand is first converted to either an integer or a floating point depending on its value. For example, the string "2" is converted to an integer, while the string "2.4" is converted to floating point. The operation then proceeds as before.

For floating-point calculations, the result is formatted to have the least number of decimal places necessary. For example, if you subtract 2.00 from 4.1, the result 2.1. Use the `FormatNumber` method to format the result to the desired number of decimal places.

Example

```
'Example of Subtraction
'InputA, InputB, and OutputA are edit controls.
OutputA = InputA - InputB
```

See Also + [add], / [divide, float], \ [divide, integer], Mod, * [multiply]

About

[*Control.*]About()

Displays information about the extension in a dialog box.

Parameters None

Return Value None

Comments None

Example

```
'Example of Control About()
BarCode1.About()
'Example of extension About()
About()
```

ABS

ABS(*x*)

Returns the absolute value of the specified number.

Parameter *x* The number for which to return the absolute value.

Return Value The absolute value of the specified number.

Comments Requires the Math extension.

Example

```
'Example of ABS(x)
Dim z
z = ABS(x)
```

ACOS

ACOS(*x*)

Calculates the arc cosine of the specified number.

Parameter *x* The number for which to calculate the arc cosine.

Return Value The arc cosine of the specified number.

Comments Applies only to the Math extension.

Example

```
'Example of ACOS(x)
Dim z
z = ACOS(x)
```

ACOSH

ACOSH(*x*)

Calculates the hyperbolic arc cosine of the specified number.

Parameter *x* The number for which to calculate the hyperbolic arc cosine.

Return Value The hyperbolic arc cosine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of ACOSH(x)
Dim z
z = ACOSH(x)
```

AddFilter

Tables(*TableName*).AddFilter(*ColumnName, Operator, Filtervalue*)

Adds a filter in a table.

Parameter

TableName	Name of a table.
ColumnName	Name of the column to add the filter condition to.
Operator	The filter operator to add. Available operators are "=", "begin", "<>", "<", ">", "<=", ">=", and "contain".
Filtervalue	The value to filter the table with.

Return Value None

Comments `AddFilter` is a method of the `Table` object.

Example

```
'Example of AddFilter
'Emps is a table.
'Salary is a columns in the Emps table.
Tables("Emps").AddFilter("Salary", ">=", "1000")
```

See Also RemoveFilter, RemoveAllFilters

AfterAppStart

Occurs immediately after an application is launched, before the first form is displayed.

Comments `AfterAppStart` is an event of the `Application` object.

Use `AfterAppStart` to perform logic that executes only once when the application starts up.

The `AfterAppStart` event is a great place to initialize global variables upon the start of your application.

See Also BeforeAppEnd

AfterChange

Occurs immediately after data in any control on a form changes and when data from a form's table is loaded into the form's controls.

Comments `AfterChange` is an event of the `Form` object.

Use `AfterChange` to perform logic that executes every time information in a form changes.

`AfterChange` fires after the controls on a form are created and loaded with data. `AfterChange` also fires if data changes in any control on a form.

Use `AfterChange` to display information that you are not storing in a table. For example, you might have an edit field to display a calculated total based on a stored price multiplied by a stored quantity. Using `AfterChange` calculates this total when the form is initially loaded with data from its table and any time a user changes the price or quantity.

See Also AfterLoad

AfterLoad

Occurs immediately after a form is loaded with data, but before the `AfterChange` event is triggered.

Comments `AfterLoad` is an event of the `Form` object.

Use `AfterLoad` to perform logic that executes every time the controls on a form are loaded with data from the form's linked table.

`AfterLoad` fires after the controls on a form are loaded with data. `AfterLoad` does not fire if there is no linked table for the form. `AfterLoad` also does not fire if there are no records in the form's linked table that satisfy the criteria of all active filters.

See Also Beep

AfterOpen

Occurs immediately after a form opens, but before the form is loaded with data from a table.

Comments `AfterOpen` is an event of the `Form` object.

Use `AfterOpen` to perform logic that executes only one time for the form. `AfterOpen` fires after a form's controls are created (drawn), but before the controls are loaded with data. `AfterOpen` only fires one time per form, regardless of the number of pages in the form.

`AfterOpen` is useful for copying global variables into Edit controls to display context information to the user.

See Also AfterLoad

AfterRecordCreate

Occurs immediately after a new record is created in the table linked to the current form.

Comments `AfterRecordCreate` is an event of the `Form` object.

Use `AfterRecordCreate` to perform logic that executes for all new records that are created in the form.

`AfterRecordCreate` fires after the record is created, but before the user enters any information in any of the controls.

Use `AfterRecordCreate` to initialize records with values.

A Jump to Form action that creates a record causes an `AfterRecordCreate` event to fire in the target form.

And [bitwise]And [logical]

Number1 And *Number2*

Performs a bitwise `And` operation between the arguments.

Parameters	*Number1*	First operand.
	Number2	Second operand.

Return Value The result of a bitwise `And` of the two operands.

Example

```
'Example of bitwise And
'InputA is an edit control.
If InputA And &H10 Then
    MsgBox("Bit 4 (mask = 10 hex) of Input A is set")
Else
    MsgBox("Bit 4 (mask = 10 hex) of Input A is clear")
EndIf
```

See Also Or [bitwise], Not [bitwise]

Condition1 And *Condition2*

Joins two conditions where both conditions must evaluate to TRUE for the expression to evaluate to TRUE.

Parameters	*Condition1*	First condition to be evaluated.
	Condition2	Second condition to be evaluated.

Return Value TRUE if *Condition1* evaluates to TRUE and *Condition2* evaluates to TRUE; FALSE if either condition evaluates to FALSE.

Comments Each condition is evaluated separately, then the `And` statement is performed. Note that `Xor`, `And`, `Or`, and `Not` only perform Boolean operations if both conditions are Boolean. Otherwise, they perform bitwise operations on their operands.

Example

```
'Example of joining conditions with And
'InputA, InputB, and OutputA are edit controls.
If InputA > 10 And InputB > 10 Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also Or [logical], Not [logical]

Asc

Asc(*Character*)

Returns the ASCII value of a character.

Parameter	*Character*	A string containing the character.

Return Value An integer corresponding to the ASCII code of the first character in the passed-in string.

See Also Chr

ASIN

ASIN(*x*)

Calculates the arc sine of the specified number.

Parameter	*x*	The number for which to calculate the arc sine.

Return Value The arc sine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of ASIN(x)
Dim z
z = ASIN(x)
```

ASINH

ASINH(*x*)

Calculates the hyperbolic arc sine of the specified number.

Parameter *x* The number for which to calculate the hyperbolic arc sine.

Return Value The hyperbolic arc sine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of ASINH(x)
Dim z
z = ASINH(x)
```

ATAN

ATAN(*x*)

Calculates the arc tangent of the specified number.

Parameter *x* The number for which to calculate the arc tangent.

Return Value The arc tangent of the specified number.

Comments Requires the Math extension.

Example

```
'Example of ATAN(x)
Dim z
z = ATAN(x)
```

ATAN2

ATAN(*y, x*)

Calculates the arc tangent of *y*/*x*.

Parameters *x* The denominator for which to calculate the arc tangent.

y The numerator for which to calculate the arc tangent.

Return Value The arc tangent of *y*/*x*.

Comments Requires the Math extension.

Example

```
'Example of ATAN2(y, x)
Dim z
z = ATAN(y, x)
```

ATANH

ATANH(*x*)

Calculates the hyperbolic arc tangent of the specified number.

Parameter *x* The number for which to calculate the hyperbolic arc tangent.

Return Value The hyperbolic arc tangent of the specified number.

Comments Requires the Math extension.

Example

```
'Example of ATANH(x)
Dim z
z = ATANH(x)
```

Backup

Tables(*TableName*).Backup(*BackupPath*)

Copies a table file to a backup folder.

Parameter	*TableName*	Name of the desired table.
	BackupPath	Path to the folder to copy the table file to.

Return Value TRUE is the backup was successful, FALSE if it failed.

Comments `Backup` is a method of the `Table` object. This function enables you to make a copy of an application data table file to a specified backup folder. The purpose of this function is to make it easier to protect against data loss by providing a simple way to copy tables to a separate folder while the application is running. You should `Commit` the table before using `Backup`. For the Palm OS platform, the backup location is the specified folder on the first available memory card. For Windows Mobile, the backup folder can be anywhere in the device's filesystem. Do not include a trailing backslash in the path. *New in Satellite Forms 8.*

Example

```
'Example of Backup method
'Back up employee table to \My Documents\Backup
Dim result
result = Tables("Emps").Backup("\My Documents\Backup")
```

See Also Search, BinarySearch

Beep

Beep(*Type*)

Issues a beep from the handheld device speaker.

Parameter	*Type*	Type of beep (a number).

Return Value None

Comments `Beep` is a method of the `App` object.

The *Type* parameter accepts the following values:

1 = Information
2 = Warning
3 = Error
4 = Startup
5 = Alarm
6 = Confirmation
7 = Click

The handheld device beeps for Information, Warnings, Errors, and Confirmation are the same.

Example

```
'Example of Beep
Beep(5)
```

See Also Tone

BeepAfter

Extension.BeepAfter(*str*)

Beeps when the scan is complete.

Parameter	*Str*	"On" or "Off"

Return Value Always TRUE for the Symbol Integrated Scanner extension. None for the Bar Code Reader extension.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of BeepAfter(str)
BarCode1.BeepAfter("On")
```

BeforeAppEnd

Occurs immediately prior to the application closing.

Comments `BeforeAppEnd` is an event of the `Application` object.

Use `BeforeAppEnd` to perform logic that executes when the application is closing.

If you Fail the `BeforeAppEnd` event, the application will not close. You can prevent unwanted exits from your application using this approach.

See Also AfterAppStart

BeforeClose

Occurs immediately before a form closes, but after the form is validated.

Comments `BeforeClose` is an event of the `Form` object.

Use `BeforeClose` to perform logic that executes one time before the form closes. Forms close when a jump to another form occurs or the application closes.

When a form closes, first `OnValidate` fires and then `BeforeClose` fires. If `OnValidate` fails, `BeforeClose` never fires and the form is not exited. `BeforeClose` only fires one time per form, regardless of the number of pages in the form.

Use `BeforeClose` to perform operations on data after the data has been validated.

Caution: Avoid setting the values of field objects and control objects during the `BeforeClose` event. Any data changes you make are not validated, creating the possibility of storing invalid data.

See Also AfterOpen, OnValidate

BeforeRecordDelete

Occurs immediately after a user attempts to delete a record, but before the record is deleted.

Comments `BeforeRecordDelete` is an event of the `Form` object.

Use `BeforeRecordDelete` to perform logic that executes before a record is deleted.

`BeforeRecordDelete` fires before a record is deleted from a form. If the script associated with `BeforeRecordDelete` fails – exits using the `FAIL` keyword the – record is not deleted.

Use `BeforeRecordDelete` to manage record deletion. For example, you can prevent the deletion of a master record, such as an order, if detail records, such as order lines, exist. A `BeforeRecordDelete` script could look for data in the detail table and fail if it finds records, informing the user to delete the detail records first.

See Also AfterRecordCreate

BI_GetBatteryPercent

BI_GetBatteryPercent

Returns current battery charge level as a percentage (0-100) of full.

Parameter *None*

Return Value Returns current battery charge level as a percentage (0-100) of full.

Comments BI_GetBatteryPercent requires the Battery Info extension.

Example

```
'Example of BI_GetBatteryInfo function
'edBattLevel is an edit control
edBattLevel = BI_GetBatteryPercent
```

See Also BI_GetPluggedIn

BI_GetPluggedIn

BI_GetPluggedIn

Returns 1 (true) if AC power is currently connected or 0 (false) if not.

Parameter *None*

Return Value Returns AC power connection status.

Comments BI_GetBatteryPercent requires the Battery Info extension.

Example

```
'Example of BI_GetPluggedIn function
'edACPower is an edit control
edACPower = Bool(BI_GetPluggedIn)
```

See Also BI_GetBatteryPercent

BinarySearch

Tables(*TableName*).BinarySearch(*ColumnName*, *Direction*, *SearchValue*, *RowNum*)

Finds an item in a sorted table.

Parameter	*TableName*	Name of the desired table.
	ColumnName	Name of the column to search.
	Direction	Sort order. TRUE for ascending; FALSE for descending.
	SearchValue	The value to search for.
	RowNum	If found, the zero-based row number of *SearchValue*.

Return Value TRUE if the method finds *SearchValue*; FALSE if it does not find *SearchValue*.

Comments `BinarySearch` is a method of the `Table` object. If the item specified by *SearchValue* is found, the fourth parameter, *RowNum*, indicates the row number of the item. If the *SearchValue* is NOT found, *RowNum* indicates the row number the item *would be* found at if it existed, also known as the *Sort Position*. The sort position of an unfound item can be used to move a new item into the correct sorted row without having to re-sort the table (see the KnowledgeBase for more details and sample script code).

Example

```
'Example of BinarySearch method
'Search table for a specific employee.
Dim RowNum, fFound

fFound = Tables("Emps").BinarySearch("Name", True,_
"John Smith", RowNum)
```

See Also Lookup, Search, InsertionSort, QuickSort

Bool

Bool(*Variable*)

Converts a variable into a Boolean value.

Parameter	*Variable*	A variable.

Return Value TRUE if the variable is non-zero; FALSE if the variable is zero

See Also Float, Int, Str

BV_Bitsize

BV_Bitsize()

Returns the current bitmap's size.

Parameters None

Return Value The current bitmap's size.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_Bitsize()
Dim x
x = BV_Bitsize()
```

BV_ColorTableSize

BV_ColorTableSize()

Returns the current bitmap's color table size.

Parameters None

Return Value The current bitmap's color table size.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_ColorTableSize()
Dim x
x = BV_ColorTableSize()
```

BV_Compress

BV_Compress()

Compresses the bitmap.

Parameters None

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_Compress()
BV_Compress()
```

BV_Create

BV_Create(*width*, *height*, *BitDepth*, *UseColorTable*)

Creates a bitmap structure.

Parameters

width	Width of the new bitmap in pixels.
height	Height of the new bitmap in pixels.
BitDepth	Color depth of the new bitmap; must be either 1, 2, 4, or 8.
UseColorTable	

Return Value TRUE if successful; FALSE if the method fails to create the bitmap.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_Create(width, height, BitDepth, UseColorTable)
Dim bCreated
bCreated = BV_Create(26, 40, 8, FALSE)
```

BV_CreateBitmapWindow

BV_CreateBitmapWindow()

Creates an offscreen bitmap window.

Parameters None

Return Value TRUE if successful; FALSE if the method fails to create the bitmap window.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_CreateBitmapWindow()
Dim bCreated
bCreated = BV_CreateBitmapWindow()
```

See Also BV_DeleteWindow

BV_DeleteWindow

BV_DeleteWindow()

Deletes the offscreen bitmap window and sets the current drawing window to the previous drawing window.

Parameters None

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_DeleteWindow()
BV_DeleteWindow()
```

See Also BV_CreateBitmapWindow

BV_DrawBitmap

BV_DrawBitmap(*X_Coord*, *Y_Coord*)

Draws the current bitmap to the screen at the position specified.

Parameters	*X_Coord*	X-origin of the bitmap in pixels.
	Y_Coord	Y-origin of the bitmap in pixels.

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_DrawBitmap(X_Coord, Y_Coord)
BV_DrawBitmap(10, 10)
```

See Also BV_PaintBitmap

BV_GetBits

BV_GetBits()

Returns the bitmap's data.

Parameters None

Return Value The bitmap's data.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_GetBits()
Dim BitmapData
BitmapData = BV_GetBits()
```

BV_GetColorTable

BV_GetColorTable()

Returns the bitmap's color table data.

Parameters None

Return Value The bitmap's color table data.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_GetColorTable()
Dim BitmapData
BitmapData = BV_GetColorTable()
```

BV_GetResPointer

BV_GetResPointer()

Passes the currently open resource's pointer to Bitmap Viewer.

Parameters None

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_GetResPointer()
```

BV_GetWinBitmap

BV_GetWinBitmap()

Returns the current drawing window as a bitmap.

Parameters None

Return Value The bitmap in the current drawing window.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_GetWinBitmap()
Bitmap2 = BV_GetWinBitmap()
```

BV_ISROM35

BV_ISROM35()

Checks to see if the current handheld has Palm OS 3.5 or greater.

Parameters None

Return Value TRUE if the handheld has Palm OS 3.5 or greater; FALSE if not.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_ISROM35()
Dim bIsROM35
bIsROM35 = BV_ISROM35()
```

BV_ModifyBitmapValue

BV_ModifyBitmapValue(*BitmapValue*,)

Changes the corresponding bitmap value and returns the previous value.

Parameters *BitmapValue*

Return Value None

Comments Applies only to the Bitmap extension. Be very careful when modifying the bitmap values as entering an improper value may crash the application.

Example

```
'Example of BV_ModifyBitmapValue(BitmapValue, ???)
PrevValue = BV_ModifyBitmapValue(BitmapValue, 29)
```

BV_PaintBitmap

BV_PaintBitmap(*X_Coord*, *Y_Coord*)

Draws the current bitmap to the screen at the position specified.

Parameters *X_Coord* X-origin of the bitmap in pixels.

Y_Coord Y-origin of the bitmap in pixels.

Return Value None

Comments Applies only to the Bitmap extension. Use this function if the bitmap has its own color table.

Example

```
'Example of BV_PaintBitmap(X_Coord, Y_Coord)
BV_PaintBitmap(10, 10)
```

See Also BV_DrawBitmap

BV_ReleaseBitmap

BV_ReleaseBitmap()

Releases the current bitmap's pointer.

Parameters None

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_ReleaseBitmap()
BV_ReleaseBitmap()
```

BV_ReturnBitmapValue

BV_ReturnBitmapValue(*BitmapValue*)

Returns the corresponding bitmap value.

Parameter *BitmapValue*

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_ReturnBitmapValue(BitmapValue)
Value = BV_ReturnBitmapValue(BVal)
```

BV_SetWindowBounds

BV_SetWindowBounds(*TopLeftX*, *TopLeftY*, *HorExtent*, *VerExtent*)

Sets the offscreen window's bounds.

Parameters *TopLeftX* X-origin of the offscreen window in pixels.

TopLeftY Y-origin of the offscreen window in pixels.

HorExtent Width of the offscreen window in pixels.

VerExtent Height of the offscreen window in pixels.

Return Value None.

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_SetWindowBounds(TopLeftX, TopLeftY, HorExtent,
VerExtent)
BV_SetWindowBounds(10, 10, 40, 80)
```

BV_SIZE

BV_SIZE()

Returns the size of the bitmap.

Parameters None

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_SIZE()
Dim x
x = BV_SIZE()
```

BV_WinSetDrawWindow

BV_WinSetDrawWindow()

Sets the offscreen window as the current drawing window.

Parameters None

Return Value None

Comments Applies only to the Bitmap extension.

Example

```
'Example of BV_WinSetDrawWindow()
BV_WinSetDrawWindow()
```

CalcTextWidth

CalcTextWidth(*string*)

Returns the width in of pixels that a given string requires, using the current font.

Parameter *String* Text string to calculate width of.

Return Value Width in pixels that String requires, using the current font.

Comments Requires the Color Graphics extension.

Example

```
'Example of CalcTextWidth function
width = CalcTextWidth("Sample Text")
```

CameraCapture

CameraCapture(*strPath, strFilename, strWindowTitle, quality, videotype, width, height, videotimelimit, capturemode*)

[Windows Mobile 5 or higher Pocket PC only.] Capture photo or video from camera.

Parameter	*strPath*	Path of folder to save photo/video file into, ending with "\".
	strFilename	Name of photo or video file including proper suffix (.JPG).
	strWindowTitle	Title of camera capture window.
	quality	Photo quality. 0 = Default, 1 = Low, 2 = Normal, 3 = High.
	videotype	Video type. 1 = Standard, 2 = Messaging, 0 = All.
	width	Pixel width of image or video.
	height	Pixel height of image or video.
	videotimelimit	Max time in seconds of video to capture.
	capturemode	Capture mode. 0 = Photo, 1 = Video only, 2 = Video + Audio.

Return Value Result code of 0 for success (no error), 1 indicates user canceled the capture, -2147024882 indicates insufficient memory to save the photo/video, -2147024809 indicates an invalid argument was supplied.

Comments Requires the WM5Camera extension.

Example *See the WM5Camera sample project.*

CanClose

Forms(*FormName*).CanClose

Returns or sets the CanClose property of a form.

Paramete *FormName* Name of a form.

Return Value True if the X/OK button is visible on this form, False if it is not visible, when used to access the property, or none when used to set the property.

Comments `CanClose` is a property of the `Form` object, and is applicable to the Pocket PC platform only. This feature is ignored for Palm. Set this property False to hide the X/OK button.

Example

```
'Example of a CanClose property
If Forms().CanClose = True then
    MsgBox("The X/OK button is visible.")
Else
    MsgBox("Let's hide the X/OK button.")
    'set the X/OK button hidden
    Forms().CanClose = False
EndIf
```

Caption

Control(*ControlName*).Caption

Returns or sets the text label associated with a control.

Paramete *ControlName* Name of a control.

Return Value String containing the caption of the specified control, if used to access the caption, or none if used to set the caption.

Comments `Caption` is a property of the `Control` object. Use to change the Text property of Title, Text, Check Box, Radio Button, and Button controls.

Example

```
'Example of a Caption property
'OutputA is an edit control.
'Checkbox1 is a check box.
OutputA = InputA.Caption
CheckBox1.Caption = "New Caption"
```

CBRT

CBRT(*x*)

Calculates the cube root of the specified number.

Parameter *x* The number for which to calculate the cube root.

Return Value The cube root of the specified number.

Comments Requires the Math extension.

Example

```
'Example of CBRT(x)
Dim z
z = CBRT(3)
```

Chr

Chr(*ASCII_Value*)

Returns the character associated with an ASCII value.

Parameter *ASCII_Value* An ASCII value.

Return Value Returns a string containing the character represented by the specified ASCII value.

Comments `Chr` is generally used to specify characters that cannot be entered with the keyboard, such as a new line character in a string, as shown in the following example:

Example

```
'Print a 2-line message
MsgBox("Line 1" & Chr(10) & "Line 2")
```

See Also Asc

ClosePort

ClosePort()

Closes the serial port, which saves battery power.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of ClosePort()
ClosePort()
```

See Also OpenPort

CM_Connect

CM_Connect(*strURL, timeout*)

Connect to the internet, using the specified URL to determine the best connection method.

Parameters

strURL	URL to use for determining best connection method.
timeout	Time in milliseconds to wait before giving up on connection.

Return Value Returns a numeric value indicating the status of the connection request.

Comments Requires the ConnectionMgr extension for Windows Mobile (not supported by the Windows CE OS). Result codes include:

-1 = Connection Manager Not Supported on Device

0 = CONNMGR_STATUS_UNKNOWN

16 = CONNMGR_STATUS_CONNECTED

32 = CONNMGR_STATUS_DISCONNECTED

33 = CONNMGR_STATUS_CONNECTIONFAILED

34 = CONNMGR_STATUS_CONNECTIONCANCELED

35 = CONNMGR_STATUS_CONNECTIONDISABLED

36 = CONNMGR_STATUS_NOPATHTODESTINATION

37 = CONNMGR_STATUS_WAITINGFORPATH

38 = CONNMGR_STATUS_WAITINGFORPHONE

64 = CONNMGR_STATUS_WAITINGCONNECTION

65 = CONNMGR_STATUS_WAITINGFORRESOURCE

66 = CONNMGR_STATUS_WAITINGFORNETWORK

128 = CONNMGR_STATUS_WAITINGDISCONNECTION

129 = CONNMGR_STATUS_WAITINGCONNECTIONABORT

Example

```
'Example of CM_Connect
edResult = CM_Connect( edURL, edTimeout )
```

See Also CM_ConnectByIndex, CM_Disconnect, CM_GetConnectionName, CM_HasConnectionMgr

CM_ConnectByIndex

CM_ConnectByIndex(*index, timeout*)

Connect to the internet using the specified connection method.

Parameters *index* Index of the desired connection method, retrieved via the CM_GetConnectionName function.

timeout Time in milliseconds to wait before giving up on connection.

Return Value Returns a numeric value indicating the status of the connection request.

Comments Requires the ConnectionMgr extension for Windows Mobile (not supported by the Windows CE OS). Result codes include:

-1 = Connection Manager Not Supported on Device

0 = CONNMGR_STATUS_UNKNOWN

16 = CONNMGR_STATUS_CONNECTED

32 = CONNMGR_STATUS_DISCONNECTED

33 = CONNMGR_STATUS_CONNECTIONFAILED

34 = CONNMGR_STATUS_CONNECTIONCANCELED

35 = CONNMGR_STATUS_CONNECTIONDISABLED

36 = CONNMGR_STATUS_NOPATHTODESTINATION

37 = CONNMGR_STATUS_WAITINGFORPATH

38 = CONNMGR_STATUS_WAITINGFORPHONE

64 = CONNMGR_STATUS_WAITINGCONNECTION

65 = CONNMGR_STATUS_WAITINGFORRESOURCE

66 = CONNMGR_STATUS_WAITINGFORNETWORK

128 = CONNMGR_STATUS_WAITINGDISCONNECTION

129 = CONNMGR_STATUS_WAITINGCONNECTIONABORT

Example

```
'Example of CM_ConnectByIndex
edResult = CM_ConnectByIndex( edConnIndex, edTimeout )
```

See Also CM_Connect, CM_Disconnect, CM_GetConnectionName, CM_HasConnectionMgr

CM_Disconnect

CM_Disonnect()

Inform the OS that we're done with the current connection and it may now be disconnected.

Parameters *None*

Return Value Returns a numeric value indicating the status of the disconnection request.

Comments Requires the ConnectionMgr extension for Windows Mobile (not supported by the Windows CE OS). The Windows Mobile OS may not close the connection if it is in use by other clients/services. A result code of 0 indicates the disconnection request was successful (although the connection may not have been closed by the OS). A result code of -1 indicates that there was not an active connection started with CM_Connect or CM_ConnectByIndex to disconnect.

Example

```
'Example of CM_Disconnect
edResult = CM_Disconnect()
```

See Also CM_Connect, CM_ConnectByIndex, CM_GetConnectionName, CM_HasConnectionMgr

CM_GetConnectionName

CM_GetConnectionName(*index*)

Obtain the name of a connection method, specified by index.

Parameters *Index* Index number of the connection method to retrieve the name of.

Return Value Returns the connection name for the specified connection index.

Comments Requires the ConnectionMgr extension for Windows Mobile (not supported by the Windows CE OS). Use this function to obtain a list of all the connection methods available on the device. Start by passing an index of 0, and repeat calling the function with an increased index number until the returned string is blank. Use the connection index number to specify a desired connection with the CM_ConnectByIndex function.

Example

```
'Example of CM_GetConnectionName
'list all connections in the edConnName paragraph control
dim connidx, connname, done
connidx = 0
done = false
while not done
      connname = CM_GetConnectionName( connidx )
      if connname = "" then
            done = true
      else
            edConnName = edConnName &connidx &" - " &connname
&chr(10)
            connidx = connidx + 1
      endif
wend
```

See Also CM_Connect, CM_ConnectByIndex, CM_Disconnect, CM_HasConnectionMgr

CM_HasConnectionMgr

CM_HasConnectionMgr()

Return whether the device has the ConnectionMgr system API library, to know if we can use ConnectionMgr functions or not.

Parameters *None*

Return Value Return True if device has ConnectionMgr library, or False if device does not have ConnectionMgr.

Comments Requires the ConnectionMgr extension for Windows Mobile (not supported by the Windows CE OS). Many WinCE devices do not have the ConnectionMgr library and so the functions cannot be used on those devices. This function allows you to determine at runtime if you can call the ConnectionMgr functions on the current device or not. Call this function before attempting to use any other ConnectionMgr functions.

Example

```
'Example of CM_HasConnectionMgr
edResult = "Device Has CM: " &Bool(CM_HasConnectionMgr)
```

See Also CM_Connect, CM_ConnectByIndex, CM_Disconnect, CM_GetConnectionName

Colorize

Colorize(*enable*)

Set True to use color controls with the colors you have defined, or False to use standard system colors.

Parameters	*enable*	True to enable custom colors or False to use system colors.

Return Value None

Comments Requires the Colorizer extension. Windows Mobile platform only (ignored on Palm OS platform). On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called. To use the standard system color scheme, call Colorize(false).

Example

```
'Enable custom colors
Colorize(true)
```

See Also

ColorizeButton

ColorizeButton(*forecolor, backcolor*)

Set Button controls foreground and background colors.

Parameters	*forecolor*	Button control foreground color as hexadecimal RGB value using &Hbbggrr format.
	backcolor	Button control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of all button controls. On Palm OS, the button foreground color set here is also used for the foreground color of the checkbox, radio, droplist, and text controls, and the background color is also used for listbox controls and popped up droplists. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeButton
ColorizeButton(&HFF0000, &H00FFFF)
```

See Also

ColorizeCheckbox

ColorizeCheckbox(*forecolor, backcolor*)

Set Checkbox controls foreground and background colors.

Parameters	*forecolor*	Checkbox control foreground color as hexadecimal RGB value using &Hbbggrr format.
	backcolor	Checkbox control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of all Checkbox controls. On Palm OS, the Checkbox foreground color set here is also used for the foreground color of the button, radio, droplist, and text controls, and the background color is also used for listbox controls and popped up droplists. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeCheckbox
ColorizeCheckbox(&HFF0000, &H00FFFF)
```

See Also

ColorizeDroplist

ColorizeDroplist(*forecolor, backcolor*)

Set Droplist controls foreground and background colors.

Parameters *forecolor* Droplist control foreground color as hexadecimal RGB value using &Hbbggrr format.

backcolor Droplist control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of all Droplist controls. On Palm OS, the Droplist foreground color set here is also used for the foreground color of the button, checkbox, radio, and text controls, and the background color is also used for listbox controls. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeDroplist
ColorizeDroplist(&HFF0000, &H00FFFF)
```

See Also

ColorizeEdit

ColorizeEdit(*forecolor, backcolor*)

Set Edit controls foreground and background colors.

Parameters *forecolor* Edit control foreground color as hexadecimal RGB value using &Hbbggrr format.

backcolor Edit control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of all Edit controls. On Palm OS, the edit control foreground color set here is also used for the foreground color of paragraph controls. The background color is ignored on Palm OS, as it uses the form’s background color for edit & paragraph controls. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeEdit
ColorizeEdit(&HFF0000, &H00FF00)
```

See Also

ColorizeExtra

ColorizeExtra(*UIcolortype, color*)

Set extra Palm OS UI colors not handled in any other Colorizer functions.

Parameters *UIcolortype* The index of the UI element to set the color of.

color The color of that UI element as a hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. Palm OS platform only. This method sets the color of any UI element, as per the list of elements defined in the Palm OS. This method can be used to set control and form colors, and also element colors that are not settable using the other Colorizer functions. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. Refer to the list of Palm OS UI elements.

Example

```
'Example of ColorizeExtra
'Set color of menu text
ColorizeExtraColor(7, &H00FF00)
```

See Also

ColorizeForm

ColorizeForm(*backcolor*)

Set Form background color.

Parameters *backcolor* Form foreground color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the background color of all forms. On Palm OS, the form background color set here is also used for the background color of checkbox, radio, and text controls. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeForm
ColorizeForm(&HFF0000)
```

See Also

ColorizeInk

ColorizeInk(*forecolor, backcolor*)

Set Ink controls foreground and background colors.

Parameters *forecolor* Ink control foreground color as hexadecimal RGB value using &Hbbggrr format.

backcolor Ink control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of ink controls. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeInk
ColorizeInk(&HFF0000, &H00FFFF)
```

See Also

ColorizeListbox

ColorizeListbox(*forecolor, backcolor*)

Set Listbox controls foreground and background colors.

Parameters *forecolor* Listbox control foreground color as hexadecimal RGB value using &Hbbggrr format.

	backcolor	Listbox control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of Listbox controls. On Palm OS, the foreground color set here is also used for the foreground color of the checkbox, radio, and text controls, and the background color is also used for popped up droplists. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeListbox
ColorizeListbox(&HFF0000, &H00FFFF)
```

See Also

ColorizeLookup

ColorizeLookup(*forecolor, backcolor*)

Set Lookup controls foreground and background colors.

Parameters	*forecolor*	Lookup control foreground color as hexadecimal RGB value using &Hbbggrr format.
	backcolor	Lookup control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of Lookup controls. On Palm OS, the foreground color set here is also used for the foreground color of the checkbox, radio, and text controls, and the background color is also used for listbox controls and popped up droplists. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeLookup
ColorizeLookup(&HFF0000, &H00FFFF)
```

See Also

ColorizeParagraph

ColorizeParagraph(*forecolor, backcolor*)

Set Paragraph controls foreground and background colors.

Parameters	*forecolor*	Paragraph control foreground color as hexadecimal RGB value using &Hbbggrr format.
	backcolor	Paragraph control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of Paragraph controls. On Palm OS, this function also affects edit controls. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeParagraph
ColorizeParagraph(&HFF0000, &H00FFFF)
```

See Also

ColorizeRadio

ColorizeRadio(*forecolor, backcolor*)

Set Radio controls foreground and background colors.

Parameters *forecolor* Radio control foreground color as hexadecimal RGB value using &Hbbggrr format.

backcolor Radio control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of Radio button controls. On Palm OS, the foreground color set here is also used for the foreground color of the checkbox, button, and text controls, and the background color is also used for listbox controls and popped up droplists. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeRadio
ColorizeRadio(&HFF0000, &H00FFFF)
```

See Also

ColorizeText

ColorizeText(*forecolor, backcolor*)

Set Text controls foreground and background colors.

Parameters *forecolor* Text control foreground color as hexadecimal RGB value using &Hbbggrr format.

backcolor Text control background color as hexadecimal RGB value using &Hbbggrr format.

Return Value None

Comments Requires the Colorizer extension. This method sets the foreground & background colors of Text controls. On Palm OS, the foreground color set here is also used for the foreground color of the checkbox, radio, and button controls, and the background color is also used for listbox controls and popped up droplists. The color is specified as an RGB value in hexadecimal format, as &Hbbggrr, where bb is the blue level, gg is the green level, and rr is the red level. On the Windows Mobile platform, color changes take effect after the Colorize(true) method is called.

Example

```
'Example of ColorizeText
ColorizeText(&HFF0000, &H00FFFF)
```

See Also

CommitData

Tables(*TableName*).CommitData

Commits (saves) the cached table to storage immediately.

Parameter *TableName* Name of a table.

Return Value None

Comments `CommitData` is a method of the `Table` object. Use `CommitData` to commit the cached table data to storage immediately.

Table data is cached when it is in use, and committed to storage automatically when the application closes. If the device is reset or power is lost before the application is closed, modified data in the cache is not written to storage, and is therefore lost when the device restarts. The `CommitData` method enables you to commit cached table data to storage immediately, in order to protect against data loss from a device reset.

Note that the `CommitData` method is NOT affected by active filters. All records in a table will be committed regardless of whether they are currently filtered out of view.

Example

```
'Example of CommitData method.
'Emps is a table.
'Add new employee and commit table to storage.
Tables("Emps").CreateRecord
Tables("Emps").MoveLast
Tables("Emps").Fields("Fname") = "Joe"
Tables("Emps").Fields("Lname") = "User"
Tables("Emps").CommitData
```

See Also CreateRecord, DeleteRecord, MoveCurrent, MoveFirst, MoveLast, MovePrevious, RecordValid

COPYSIGN

COPYSIGN(*x, y*)

Returns *x* with the sign of *y*.

Parameters *x* The number receiving the sign from *y*.

y The number from which to copy the sign.

Return Value *x* with the sign of *y*.

Comments Requires the Math extension.

Example

```
'Example of COPYSIGN(x, y)
COPYSIGN(x, y)
```

COS

COS(*x*)

Calculates the cosine of the specified number.

Parameter *x* The number for which to calculate the cosine.

Return Value The cosine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of COS(x)
Dim z
z = COS(x)
```

COSH

COSH(*x*)

Calculates the hyperbolic cosine of the specified number.

Parameter *x* The number for which to calculate the hyperbolic cosine.

Return Value The hyperbolic cosine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of COSH(x)
Dim z
z = COSH(x)
```

Count

Object.Count

Returns the number of child objects in an object collection.

Parameters None

Return Value Number of child objects in the specified object collection.

Comments `Count` is a read-only property of the `Controls` collection, the `Extensions` collection, the `Fields` collection, the `Forms` collection, the `Table` object, and the `Tables` collection.

Use `Count` to determine the number of controls in a form, the number of columns in a table, the number of forms in an application, the number of records in a table, or the number of tables in an application.

Note that the `Count` property of the `Tables` collection returns the number of tables in the application. The `Count` property of the `Table` object returns the number of records in the table.

Example

```
'Example of Count property
'OutputA through OutputE are edit controls.
'Emps is a table.
'Number of forms in current application
OutputB = Forms.Count
'Number of controls in current form
OutputA = Controls.Count
'Number of tables in current application
OutputC = Tables.Count
'Number of fields in current table
OutputD = Tables().Fields.Count
'Number of records in current table
OutputE = Tables().Count
```

See Also Sum

CreateRecord

Tables(*TableName*).CreateRecord

Creates a record in a table.

Parameter *TableName* Name of a table.

Return Value None

Comments `CreateRecord` is a method of the `Table` object. This method does not prompt the user to confirm the record creation.

Example

```
'Example of CreateRecord
'Emps is a table.
'Empno, Name, and Salary are columns in the
'Emps table.
Tables("Emps").CreateRecord
Tables("Emps").MoveLast
Tables("Emps").Fields("Empno") = "150"
Tables("Emps").Fields("Name") = "Smith"
Tables("Emps").Fields("Salary") = "1000"
```

CtrlAdv

Extension.CtrlAdv(*str*)

Turns the CTRLADV property on or off.

Parameter *Str* “On” or “Off”

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of CtrlAdv(str)
BarCode1.CtrlAdv("On")
```

CurrentPage

Object.CurrentPage

Returns the zero-based number of the current page on a form.

Parameter *Object* An object.

Return Value Zero-based number of current page.

Comments `CurrentPage` is a property of the `Form` object. Zero-based numbering means that the first page is page zero.

Example

```
'Example of CurrentPage property
'OutputA is an edit control
'on the first page.
'Output2A is an edit control
'on the second page.
If Forms().CurrentPage = 0 Then
    OutputA = "First Page"
ElseIf Forms().CurrentPage = 1 Then
    Output2A = "Second Page"
EndIf
```

CurrentRecord

Forms(*FormName*).CurrentRecord

Returns the zero-based number of the current record of a form.

Parameter *FormName* Name of a form.

Return Value Zero-based number of the current record of a form.

Comments `CurrentRecord` is a property of the `Form` object.

Example

```
'Example of CurrentRecord property
'OutputA is an edit control.
'Display the number of the current record.
OutputA = Forms().CurrentRecord
```

Data

Object.Data

Accesses data from an object or assigns data to the object.

Parameter *Object* An object.

Return Value None for assignments.
Data stored in the object for accesses.

Comments `Data` is a property of the `Control` and `Field` objects.

Use the `Data` property to access data from controls and fields, or to assign data to controls and fields.

Example

```
'Example of the Data property for both assignmentand access
'InputA and OutputA through OutputD are edit controls.
'Emps is a table.
'Data from a control
OutputA.Data = InputA.Data
'Data from a field in a table
'for the current record
OutputB.Data = Tables("Emps").Fields("Salary").Data
'The Data property is the default property for
'controls. Thus, the following statement is
'equivalent to the one above:
OutputC = InputA
OutputD = Tables("Emps").Fields("Salary")
```

DateToSysDate

DateToSysDate(*Date*)

Converts a user-readable date to days since January 1, 1904.

Parameter *Date* Date in the format specified in the handheld preferences.

Return Value Days since January 1, 1904.

Comments `DateToSysDate` is a method of the `App` object.

See Also SysDateToDate, GetSysDate, GetSysTime, SysTimeToTime, TimeToSysTime

Delay

Delay(*Duration*)

Waits a specified number of milliseconds.

Parameter *Duration* Duration of the delay in milliseconds.

Return Value None

Comments `Delay` is a method of the `App` object.

Example

```
'Example of Delay method
'InputA and InputB are edit controls.
Tone(InputA, InputB)
Delay(1000)
Tone(2 * InputA, InputB)
```

DeleteRecord

Tables(*TableName*).DeleteRecord(*RecordNumber*)

Deletes a record from a table. This method does not prompt the user to confirm deletion.

Parameters	*TableName*	Name of a table.
	RecordNumber	Zero-based number of the row to be deleted.

Return Value None

Comments `DeleteRecord` is a method of the `Table` object.

Example

```
'Example of Delete Record 'method
Dim count, i
'Get number of records in the table
count = Tables("emp").Count()
'Iterate through all records and delete
'the first record in the table.
for i=0 to count-1
    Tables("emp").DeleteRecord(0)
next i
```

See Also CreateRecord

Dim

Dim *Variable*

Dimensions (declares) a variable as `VARIANT` type.

Parameter *Variable* Name of the variable.

Return Value None

Comments Variables of type `VARIANT` are initialized to a specific data type when you assign them a value. For example, assigning the value “My text” to a `VARIANT` variable changes the variable’s data type to `STRING`. The data types for variables are:

`FLOAT`:	floating-point number
`INT`:	integer
`STRING`:	string

Use the conversion operators `Float`, `Int`, and `Str` to convert one data type to another.

Example

```
'Example of Dimensioning variables
Dim x
Dim y
Dim z
'Assign variables to values.
x  = 33      ' x becomes INT
y  = 33.00   ' y becomes FLOAT
z  = "33"    ' z becomes STRING
```

See Also Data type conversion operators: Float, Int, Str

DisableScanner

Extension.DisableScanner()

Turns off the serial port and disables scanning. Uses less battery power.

Parameters None

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Caution: DO Not Call Other Symbol Scanner Functions While the Symbol Integrated Scanner scanner is disabled. A fatal exception may result.

Example

```
'Example of DisableScanner()
BarCode1.DisableScanner()
```

DrawBar

DrawBar(*X*, *Y*, *W*, *H*)

Draws a filled bar, as for a bar graph, using the current Pen and Fill colors.

Parameters	*X*	X-origin of the bottom-left corner of the bar.
	Y	Y-origin of the bottom-left corner of the bar.
	W	Width of the bar.
	H	Height of the bar.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of DrawBar(X, Y, W, H)
DrawBar(20, 20, 10, 27)
```

See Also EraseBar

DrawCircle

DrawCircle(*X0*, *Y0*, *R*)

Draws a circle using the current Pen and Fill colors.

Parameters	*X0*	X-origin of the center of the circle in pixels.
	Y0	Y-origin of the center of the circle in pixels.
	R	Radius of the circle in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of DrawCircle(X0, Y0, R)
DrawCircle(35, 15, 20)
```

See Also EraseCircle

DrawLine

DrawLine(*X0*, *Y0*, *X1*, *Y1*)

Draws a line using the current Pen color.

Parameters *X0* X-origin of the line in pixels.
Y0 Y-origin of the line in pixels.
X1 X-end of the line in pixels.
Y1 Y-end of the line in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
‘Example of DrawLine(X0, Y0, X1, Y1)
DrawLine(10, 10, 40, 40)
```

See Also EraseLine

DrawRect

DrawRect(*X0*, *Y0*, *X1*, *Y1*)

Draws a rectangle using the current Pen and Fill colors.

Parameters *X0* X-origin of the upper left corner of the rectangle in pixels.
Y0 Y-origin of the upper left corner of the rectangle in pixels.
X1 X-origin of the lower right corner of the rectangle in pixels.
Y1 Y-origin of the lower right corner of the rectangle in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
‘Example of DrawRect(X0, Y0, X1, Y1)
DrawRect(10, 10, 40, 40)
```

See Also EraseRect

DrawRoundedRect

DrawRoundedRect(*X0*, *Y0*, *X1*, *Y1*, *R*)

Draws a rounded rectangle with the specified corner radius using the current Pen and Fill colors.

Parameters *X0* X-origin of the upper left corner of the rectangle in pixels.
Y0 Y-origin of the upper left corner of the rectangle in pixels.
X1 X-origin of the lower right corner of the rectangle in pixels.
Y1 Y-origin of the lower right corner of the rectangle in pixels.
R Radius of the rectangle’s corners.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
‘Example of DrawRoundedRect(X0, Y0, X1, Y1, R)
DrawRoundedRect(10, 10, 40, 40, 15)
```

See Also EraseRect

DrawText

DrawText(*Strg*, *X*, *Y*)

Draws text at the specified point using the current text color.

Parameters	*Strg*	Text to draw.
	X	Y-origin of the upper left corner of the text in pixels.
	Y	X-origin of the upper left corner of the text in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of DrawText(Strg, X, Y)
DrawText("I'm here", 40, 40)
```

See Also EraseText, InvertText

DRand48

DRand48()

Returns a psuedo-random number greater than 0 but less than or equal to 1.

Parameters None

Return Value A psuedo-random number greater than 0 but less than or equal to 1.

Comments Requires the Random Number Generator extension.

Example

```
'Example of DRand48()
Dim z
z = DRand48()
```

DREM

DREM(*y, x*)

Calculates the remainder of *x*/*y*.

Parameters	*x*	The numerator of the division operation.
	y	The denominator of the division operation.

Return Value The remaindert of *x*/*y*.

Comments Requires the Math extension.

Example

```
'Example of DREM(x, y)
Dim z
z = DREM(x, y)
```

EditExAbout

EditExAbout()

Displays information about the extension in a dialog box.

Parameters None

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExAbout()
EditExAbout()
```

EditExAppendText

EditExAppendText(*AppDesIndex*, *Text*)

Appends text to the specified control's text.

Parameters	*AppDesIndex*	Index of the desired Edit or Paragraph control on the form.
	Text	The text to append to the text in the specified control.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExAppendText(AppDesIndex, Text)
EditExAppendText(2, "More text")
```

EditExBackspace

EditExBackspace(*AppDesIndex*)

Backspaces one character in the specified control.

Parameter *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExBackspace(AppDesIndex)
EditExBackspace(2)
```

EditExDeleteText

EditExDeleteText(*AppDesIndex*, *Start*, *End*)

Deletes the specified text from the specified control.

Parameters	*AppDesIndex*	Index of the desired Edit or Paragraph control on the form.
	Start	Start position from which to begin deleting text in the specified control.
	End	End position at which to stop deleting text in the specified control.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExDeleteText(AppDesIndex, Start, End)
EditExDeleteText(2, 4, 10)
```

EditExGetInsertion

EditExGetInsertion(*AppDesIndex*)

Returns the current insertion position in the specified control.

Parameter *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Return Value Integer indicating the position of the insertion point in the specified control.

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExGetInsertion(AppDesIndex)
Dim iPoint
iPoint = EditExGetInsertion(2)
```

EditExGetSelectionEnd

EditExGetSelectionEnd(*AppDesIndex*)

Returns the end position of the text selection in the specified control.

Parameter *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Return Value Integer indicating the end position of the selected text in the specified control.

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExGetSelectionEnd(AppDesIndex)
Dim iSelEnd
iSelEnd = EditExGetSelectionEnd(2)
```

EditExGetSelectionStart

EditExGetSelectionStart(*AppDesIndex*)

Returns the start position of the text selection in the specified control.

Parameter *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Return Value Integer indicating the start position of the selected text in the specified control.

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExGetSelectionStart(AppDesIndex)
Dim iSelEnd
iSelEnd = EditExGetSelectionStart(2)
```

EditExInsertText

EditExInsertText(*AppDesIndex*, *Text*)

Inserts the specified text at the current insertion position in the specified control.

Parameters *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Text Text to insert into the specified control.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExInsertText(AppDesIndex, Text)
EditExInsertText(2, "Some text")
```

EditExInStr

EditExInStr(*Start*, *String1*, *String2*, *Compare*)

Returns the position of the first occurrence of one string within another.

Parameters *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

String1 The string to search in.

String2 String containing the text to search for.

Compare 0 = Binary comparison. 1 = case-insensitive text comparison. Default setting is 0.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExInStr(Start, String1, String2, Compare)
EditExInStr(5, sText1, sText2, 1)
```

EditExSetInsertion

EditExSetInsertion(*AppDesIndex*, Insertion)

Sets the insertion position in the specified control.

Parameters *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Insertion Index at which to set the insertion point in the specified control.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExSetInsertion(AppDesIndex, Insertion)
EditExSetInsertion(2, 5)
```

EditExSetSelection

EditExSetSelection(*AppDesIndex*, *Start*, *End*)

Sets the text selection in the specified control.

Parameters *AppDesIndex* Index of the desired Edit or Paragraph control on the form.

Start Start position at which to set the text selection in the specified control.

End End position at which to stop the text selection in the specified control.

Return Value None

Comments Applies only to the EditEx extension.

Example

```
'Example of EditExSetSelection(AppDesIndex, Start, End)
EditExSetSelection(2, 4, 10)
```

Empty

x = Empty

Sets a table field or variable to an Empty state.

Parameters None

Return Value None

Comments Applies only to table fields and to variables, but not to controls.

Fields in a new record are always initialized to Empty, which is not equal to "" (empty string) or 0. New variables are also initialized to Empty. Once you assign a value to a field or variable, it is no longer Empty. However, you can make the table field or variable empty again by assigning it to Empty. You can test whether a field or variable is empty using the IsEmpty(object) keyword.

Caution: Empty and IsEmpty *must not* be used with controls. Note: AppDesigner does not currently enforce this rule when compiling your application, so you need to make sure not to use Empty with controls!

Example

```
'Example of assigning Empty to a var and testing with IsEmpty
x = Empty
if IsEmpty(x) then msgbox("x is Empty!")

'Example of assigning Empty to a table field
Tables(tablename).Fields(fieldname) = Empty
```

See Also IsEmpty

EnableScanner

Extension.EnableScanner()

Turns on the serial port and disables scanning. Uses more battery power.

Parameters None

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Caution: You must call this method before calling other Symbol Integrated Scanner methods. Otherwise, a fatal exception may result.

Example

```
'Example of EnableScanner()
BarCode1.EnableScanner()
```

EraseBar

EraseBar(*X*, *Y*, *W*, *H*)

Erases a filled bar drawn using the the `DrawBar` method.

Parameters

Parameters	*X*	X-origin of the bottom-left corner of the bar.
	Y	Y-origin of the bottom-left corner of the bar.
	W	Width of the bar.
	H	Height of the bar.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of EraseBar(X, Y, W, H)
EraseBar(20, 20, 10, 27)
```

See Also DrawBar

EraseCircle

EraseCircle(*X0*, *Y0*, *R*)

Erases a circle drawn using the the `DrawCircle` method.

Parameters	*X0*	X-origin of the center of the circle in pixels.
	Y0	Y-origin of the center of the circle in pixels.
	R	Radius of the circle in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of EraseCircle(X0, Y0, R)
EraseCircle(35, 15, 20)
```

See Also DrawCircle

EraseLine

EraseLine(*X0*, *Y0*, *X1*, *Y1*)

Erases a line drawn using the the `DrawLine` method.

Parameters	*X0*	X-origin of the line in pixels.
	Y0	Y-origin of the line in pixels.
	X1	X-end of the line in pixels.
	Y1	Y-end of the line in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of EraseLine(X0, Y0, X1, Y1)
EraseLine(10, 10, 40, 40)
```

See Also DrawLine

EraseRect

EraseRect(*X0*, *Y0*, *X1*, *Y1*)

Erases a rectangle drawn using the the `DrawRect` method.

Parameters	*X0*	X-origin of the upper left corner of the rectangle in pixels.
	Y0	Y-origin of the upper left corner of the rectangle in pixels.
	X1	X-origin of the lower right corner of the rectangle in pixels.
	Y1	Y-origin of the lower right corner of the rectangle in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions. If the current Fill color is Transparent, this method does not erase the Fill area of the specified rectangle.

Example

```
'Example of EraseRect(X0, Y0, X1, Y1)
EraseRect(10, 10, 40, 40)
```

See Also DrawRect, DrawRoundedRect

EraseText

EraseText(*Strg*, *X*, *Y*)

Draws text at the specified point using the current text color.

Parameters	*Strg*	Text to erase.
	X	Y-origin of the upper left corner of the text in pixels.
	Y	X-origin of the upper left corner of the text in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of EraseText(Strg, X, Y)
EraseText("I'm here", 40, 40)
```

See Also DrawText

ExecAction

Object.ExecAction

Executes the action associated with an object.

Parameter	*Object*	An object.

Return Value None

Comments `ExecAction` is a method of the `Control` object. You can use `ExecAction` to implement a simple form of subroutine calls. Parameters, however, are not supported and must be passed in globals.

Note: do not nest too many levels of `ExecAction` commands; Palm OS handhelds have very limited stack space.

Example

```
'Example of executing the action of a button
'Button1 and Button2 are buttons.
Button1.ExecAction
```

Exit

Exit

Exits from a script with success.

Parameters None

Return Value None

Comments Use `Exit` and `Fail` in validation events to make validation succeed or fail. Use them in `BeforeRecordDelete` to allow or prevent deletion of a record. Using `Exit` or `Fail` in other events simply ends the current script.

Example

```
'Example of the Exit and Fail keywords.
'Use this code in a validation event.
'InputA is an edit control.
'Entering a number less than 10 in InputA fails validation.
If InputA >= 10 Then
    Exit
Else
    Fail
EndIf
```

See Also Fail

EXP

EXP(*x*)

Calculates the exponential *e* to the *x*.

Parameter *x* The number to raise to the exponential of itself.

Return Value The exponential *e* to the *x*.

Comments Requires the Math extension.

Example

```
'Example of EXP(x)
Dim z
Dim x
x = 4.0
‘Calculate 4.0 to the fourth power
z = EXP(x)
```

EXPM1

EXPM1(*x*)

Calculates the exponential *e* to the *x* -1.

Parameter *x* The number to raise to the exponential of itself.

Return Value The exponential *e* to the *x* -1.

Comments Requires the Math extension.

Example

```
'Example of EXPM1(x)
Dim z
Dim x
x = 4.0
z = EXPM1(x)
```

Fail

Fail

Exits from a routine with failure.

Parameters None

Return Value None

Comments Use `Exit` and `Fail` in validation events to make validation succeed or fail. Use them in `BeforeRecordDelete` to allow or prevent deletion of a record. Using `Exit` or `Fail` in other events simply ends the current script.

Example

```
'Example of the Exit and Fail keywords
'Use this code in a validation event.
'InputA is an edit control.
'Entering a number less than 10 in InputA fails validation.
If InputA >= 10 Then
    Exit
Else
    Fail
EndIf
```

See Also Exit

FastTrack

Extension.FastTrack(*boolean*)

Specifies how often the `OnClick` event fires.

Parameter *boolean*
0 = Fire OnClick event every time the stylus moves.
1 = Fire OnClick event only after the user lifts the stylus.
The default setting is 0.

Return Value None

Comments Applies only to the Slider and Color Slider controls.

Example

```
‘Example of FastTrack(boolean)
ColorSlider1.FastTrack(1)
```

FF_DeviceHasVFS

FF_DeviceHasVFS()

[Palm OS only.] Check whether the current device has Palm OS VFS memory card support.

Parameters *None*

Return Value Returns True if device has VFS, or False if not.

Comments FF_DeviceHasVFS requires the FindFiles extension. This function is for Palm OS platform devices only. Use this function to check if the current Palm OS device supports VFS expansion memory cards. You may wish to check this on startup of your app, and if no VFS support is provided you would then make sure that all file operations were for internal memory only.

Example

```
'Example of FF_DeviceHasVFS function
if not FF_DeviceHasVFS then
      MsgBox(“Palm device does not have memory card support”)
endif
```

See Also FF_FindFirstFileVFS, FF_FindNextFileVFS, FF_GetNextVolRef, FF_GetVFSLabel, FF_GetVFSVolRef, FF_SetVFSVolRef

FF_FindClose

FF_FindClose()

Close an open Find operation when done with it.

Parameters *None*

Return Value *None*

Comments FF_FindClose requires the FindFiles extension. Use this function to close an active Find operation that was opened with FF_FindOpen, when you are finished with it.

See Also FF_FindClose

FF_FindFirstDir

FF_FindFirstDir(*path, name*)

Find first directory in a new search.

Parameters

path String containing the starting path from which you want to start searching, which should begin and end with a backslash "\" char. [For Palm OS this operates on a VFS card volume, and for Pocket PC this parameter is ignored.]

name Name is the directory substring you wish to match. For Palm OS, leave it blank to match any directory. For Pocket PC, use an "*" asterisk wildcard to match any directory.

Return Value The function returns 1 (True) to indicate a match, or 0 (False) if no match was found.

Comments FF_FindFirstDir requires the FindFiles extension. If this function finds a match, then you would use FF_GetFileName to get the name of the matching directory that was found. This function might be called in a loop to obtain a list of directories, as in the example below.

Example

```
'Example of finding a list of directories
'This would operate on a VFS memory card on Palm OS
dim bFound, dirname, dirlist
edFound = 0
dirlist = ""
pgFileList = ""
'the path parameter is ignored on PPC
'search VFS card from specified path
bFound = FF_FindFirstDir( edpath, edname )
while bFound = true
      dirname = FF_GetFileName
      dirlist = dirlist &dirname
      edFound = edFound + 1
      bFound = FF_FindNextDir
      if bFound then dirlist = dirlist &chr(10)
wend
FF_FindClose
if edFound = 0 then
      pgFileList = "Did not find any matching dirs."
else
      pgFileList = dirlist
endif
```

See Also FF_FindFirstFile, FF_FindFirstFileVFS, FF_FindNextDir, FF_FindNextFile, FF_FindNextFileVFS

FF_FindFirstFile

FF_FindFirstFile(*name, creator, type*)

Find first file in a new search.

Parameters	*name*	Name is the filename substring you wish to match. For Palm OS, leave it blank to match any file. For Pocket PC, use an "*" asterisk wildcard to match any file.
	creator	The 4-character CreatorID string of the file to match, leave blank to match any. [Palm OS only, ignored on Pocket PC]
	type	The 4-character Type string of the file to match, leave blank to match any. [Palm OS only, ignored on Pocket PC]

Return Value The function returns 1 (True) to indicate a match, or 0 (False) if no match was found.

Comments FF_FindFirstFile requires the FindFiles extension. If this function finds a match, then you would use additional functions like FF_GetFileName, FF_GetFileSize, or FF_GetFileDateModified to get the name, size, and date of the matching file that was found. This function might be called in a loop to obtain a list of files, as in the example below.

Example

```
'find files
dim bFound, filename, filesize, filelist
edFound = 0
filelist = ""
pgFileList = ""
bFound = FF_FindFirstFile( edname, edcreator, edtype )
while bFound = true
      filename = FF_GetFileName
      filesize = FF_GetFileSize
      filelist = filelist &filename &", " &filesize &" bytes"
      edFound = edFound + 1
      bFound = FF_FindNextFile
      if bFound then filelist = filelist &chr(10)
wend
FF_FindClose
if edFound = 0 then
      pgFileList = "Did not find any matching files."
else
      pgFileList = filelist
endif
```

See Also FF_FindFirstDir, FF_FindFirstFileVFS, FF_FindNextDir, FF_FindNextFile, FF_FindNextFileVFS

FF_FindFirstFileVFS

FF_FindFirstFileVFS(*path, name, creator, type*)

[Palm OS only.] Find first file in a new search on a VFS memory card.

Parameters	*path*	Path is the starting path from which you want to start searching, which should begin and end with a backslash "\" char. [Palm OS only, ignored on Pocket PC]
	name	Name is the filename substring you wish to match. For Palm OS, leave it blank to match any file. For Pocket PC, use an "*" asterisk wildcard to match any file.
	creator	The 4-character CreatorID string of the file to match, leave blank to match any. [Palm OS only, ignored on Pocket PC]

type The 4-character Type string of the file to match, leave blank to match any. [Palm OS only, ignored on Pocket PC]

Return Value The function returns 1 (True) to indicate a match, or 0 (False) if no match was found.

Comments FF_FindFirstFileVFS requires the FindFiles extension. On Palm OS platform devices, this function is used to find files on a memory card. On Pocket PC devices, this function works identically to the FF_FindFirstFile function. If this function finds a match, then you would use additional functions like FF_GetFileName, FF_GetFileSize, or FF_GetFileDateModified to get the name, size, and date of the matching file that was found. This function might be called in a loop to obtain a list of files, as in the example below.

Example

```
'find files on a memory card
dim bFound, filename, filesize, filelist
edFound = 0
filelist = ""
pgFileList = ""
bFound = FF_FindFirstFileVFS( edpath, edname, edcreator, edtype )
while bFound = true
      filename = FF_GetFileName
      filesize = FF_GetFileSize
      filelist = filelist &filename &", " &filesize &" bytes"
      edFound = edFound + 1
      bFound = FF_FindNextFileVFS
      if bFound then filelist = filelist &chr(10)
wend
FF_FindClose
if edFound = 0 then
      pgFileList = "Did not find any matching files."
else
      pgFileList = filelist
endif
```

See Also FF_DeviceHasVFS, FF_FindNextFileVFS, FF_GetNextVolRef, FF_GetVFSLabel, FF_GetVFSVolRef, FF_SetVFSVolRef

FF_FindNextDir

FF_FindNextDir()

Find next directory in existing search started with FF_FindFirstDir.

Parameters *None*

Return Value The function returns 1 (True) to indicate a match, or 0 (False) if no match was found.

Comments FF_FindNextDir requires the FindFiles extension. If this function finds a match, then you would use FF_GetFileName to get the name of the matching directory that was found. This function might be called in a loop to obtain a list of directories, as in the example shown in the FF_FindFirstDir reference. For Palm OS devices, this function operates on internal memory, and you would use FF_FindNextDirVFS for memory cards.

Example *See example for FF_FindFirstDir.*

See Also FF_FindFirstDir, FF_FindFirstFile, FF_FindFirstFileVFS, FF_FindNextFile, FF_FindNextFileVFS

FF_FindNextFile

FF_FindNextFile()

Find next file in existing search started with FF_FindFirstFile.

Parameters *None*

Return Value The function returns 1 (True) to indicate a match, or 0 (False) if no match was found.

Comments FF_FindNextFile requires the FindFiles extension. If this function finds a match, then you would use FF_GetFileName to get the name of the matching directory that was found. This function might be called in a loop to obtain a list of directories, as in the example shown in the FF_FindFirstFile reference. For Palm OS devices, this function operates on internal memory, and you would use FF_FindNextFileVFS for memory cards.

Example *See example for FF_FindFirstFile.*

See Also FF_FindFirstDir, FF_FindFirstFile, FF_FindFirstFileVFS, FF_FindNextDir, FF_FindNextFileVFS

FF_FindNextFileVFS

FF_FindNextFileVFS()

[Palm OS only.] Find next file on memory card in search started with FF_FindFirstFileVFS.

Parameters *None*

Return Value The function returns 1 (True) to indicate a match, or 0 (False) if no match was found.

Comments FF_FindNextFileVFS requires the FindFiles extension. If this function finds a match, then you would use FF_GetFileName to get the name of the matching directory that was found. This function might be called in a loop to obtain a list of directories, as in the example shown in the FF_FindFirstFileVFS reference. For Palm OS devices, this function operates on memory cards, and you would use FF_FindNextFile for internal memory. For Pocket PC, this function works identically to FF_FindNextFile.

Example *See example for FF_FindFirstFileVFS.*

See Also FF_DeviceHasVFS, FF_FindFirstFileVFS, FF_GetNextVolRef, FF_GetVFSLabel, FF_GetVFSVolRef, FF_SetVFSVolRef

FF_GetFileAttr

FF_GetFileAttr()

Get the attributes of the currently matching file.

Parameters *None*

Return Value The function returns a numeric value which includes the file attribute bit values. The attribute bit values are dependent on the OS platform.

Comments FF_GetFileAttr requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. File attribute bit values are listed below.

Palm OS File Attributes

&H0001	Resource database (PRC), not Record database (PDB)
&H0002	Read Only
&H0004	AppInfo block is dirty
&H0008	Backup bit is set
&H0010	OK to install newer

&H0020	Reset after install
&H0040	Copy Prevention bit is set
&H0080	Stream database
&H0100	Hidden database
&H0200	Launchable Data
&H0400	Recyclable
&H0800	Bundle
&H8000	Database is Open

Pocket PC File Attributes

&H0001	FILE_ATTRIBUTE_READONLY
&H0002	FILE_ATTRIBUTE_HIDDEN
&H0004	FILE_ATTRIBUTE_SYSTEM
&H0010	FILE_ATTRIBUTE_DIRECTORY
&H0020	FILE_ATTRIBUTE_ARCHIVE
&H0040	FILE_ATTRIBUTE_INROM
&H0080	FILE_ATTRIBUTE_NORMAL
&H0100	FILE_ATTRIBUTE_TEMPORARY
&H0200	FILE_ATTRIBUTE_SPARSE_FILE
&H0400	FILE_ATTRIBUTE_REPARSE_POINT
&H0800	FILE_ATTRIBUTE_COMPRESSED
&H1000	FILE_ATTRIBUTE_ROMSTATICREF
&H2000	FILE_ATTRIBUTE_ROMMODULE

Example

```
'find files
dim bFound, filename, fileattr, filelist
edFound = 0
filelist = ""
pgFileList = ""
bFound = FF_FindFirstFile( edname, edcreator, edtype )
while bFound = true
      filename = FF_GetFileName
      fileattr = FF_GetFileAttr
      filelist = filelist &filename &", attr:" &fileattr
      edFound = edFound + 1
      bFound = FF_FindNextFile
      if bFound then filelist = filelist &chr(10)
wend
FF_FindClose
if edFound = 0 then
      pgFileList = "Did not find any matching files."
else
      pgFileList = filelist
endif
```

See Also FF_GetFileDateBackedUp, FF_GetFileDateCreated, FF_GetFileDateModified, FF_GetFileName, FF_GetFileSize

FF_GetFileCreator

FF_GetFileCreator()

[Palm OS only.] Get the Palm OS creatorID of the currently matching file.

Parameters *None*

Return Value The function returns the creatorID 4-character string.

Comments FF_GetFileCreator requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. This function is for the Palm OS platform only, and always returns a blank string on the Pocket PC platform.

Example

```
filecreator = FF_GetFileCreator
```

See Also FF_GetFileType, FF_GetFileVersion

FF_GetFileDateBackedUp

FF_GetFileDateBackedUp()

Get the “last backed up” date of the currently matching file.

Parameters *None*

Return Value The function returns the file last backed up date in system time format.

Comments FF_GetFileDateBackedUp requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. The system time value returned can be converted into a date and time.

Example

```
'get file dates (system time seconds since 00:00 Jan 1, 1904)
'system time format includes both the date and time value
'this value is a signed 32 bit integer and can be negative
'so for date we convert to a positive by adding 2 ^ 32 to
'the returned value and converting that from seconds into
'days (divide secs by 86400 secs/day)
dim UInt32MAX, filelist
UInt32MAX = 4294967296
if FF_FindFirstFile( edname, edcreator, edtype ) = true then
      filename = FF_GetFileName
      filedateBU = FF_GetFileDateBackedUp
      filedateMO = FF_GetFileDateModified
      filedateCR = FF_GetFileDateCreated
      filelist = filename _
      &" C:" &SysDateToDate((filedateCR+UInt32MAX)/86400) _
      &", M:" &SysDateToDate((filedateMO+UInt32MAX)/86400) _
      &", B:" &SysDateToDate((filedateBU+UInt32MAX)/86400)
endif
FF_FindClose
pgFileList = filelist
```

See Also FF_GetFileAttr, FF_GetFileDateCreated, FF_GetFileDateModified, FF_GetFileName, FF_GetFileSize

FF_GetFileDateCreated

FF_GetFileDateCreated()

Get the creation date of the currently matching file.

Parameters *None*

Return Value The function returns the file creation date in system time format.

Comments FF_GetFileDateCreated requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. The system time value returned can be converted into a date and time.

Example *See the example for FF_GetFileDateBackedUp*

See Also FF_GetFileAttr, FF_GetFileDateBackedUp, FF_GetFileDateModified, FF_GetFileName, FF_GetFileSize

FF_GetFileDateModified

FF_GetFileDateModified()

Get the last modified date of the currently matching file.

Parameters *None*

Return Value The function returns the file last modified date in system time format.

Comments FF_GetFileDateModified requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. The system time value returned can be converted into a date and time.

Example *See the example for FF_GetFileDateBackedUp*

See Also FF_GetFileAttr, FF_GetFileDateBackedUp, FF_GetFileDateCreated, FF_GetFileName, FF_GetFileSize

FF_GetFileName

FF_GetFileName()

Get the name of the currently matching file or directory.

Parameters *None*

Return Value String containing the name of the currently matching file or directory.

Comments FF_GetFileName requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile/FF_FindNextFile, or directory found by FF_FindFirstDir/FF_FindNextDir.

Example

```
'find files
dim bFound, filename, filesize, filelist
edFound = 0
filelist = ""
pgFileList = ""
bFound = FF_FindFirstFile( edname, edcreator, edtype )
while bFound = true
      filename = FF_GetFileName
      filesize = FF_GetFileSize
      filelist = filelist &filename &", " &filesize &" bytes"
      edFound = edFound + 1
      bFound = FF_FindNextFile
      if bFound then filelist = filelist &chr(10)
wend
FF_FindClose
if edFound = 0 then
      pgFileList = "Did not find any matching files."
else
      pgFileList = filelist
endif
```

See Also FF_GetFileAttr, FF_GetFileDateBackedUp, FF_GetFileDateCreated, FF_GetFileDateModified, FF_GetFileSize

FF_GetFileSize

FF_GetFileSize()

Get the size of the currently matching file.

Parameters *None*

Return Value Size of the currently matching file in bytes.

Comments FF_GetFileSize requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile/FF_FindNextFile, or directory found by FF_FindFirstDir/FF_FindNextDir.

Example *See the example for FF_GetFileName*

See Also FF_GetFileAttr, FF_GetFileDateBackedUp, FF_GetFileDateCreated, FF_GetFileDateModified, FF_GetFileName

FF_GetFileType

FF_GetFileType()

[Palm OS only.] Get the Palm OS type of the currently matching file.

Parameters *None*

Return Value The function returns the 4-character Palm OS type string.

Comments FF_GetFileType requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. This function is for the Palm OS platform only, and always returns a blank string on the Pocket PC platform.

Example

```
filetype = FF_GetFileType
```

See Also FF_GetFileCreator, FF_GetFileVersion

FF_GetFileVersion

FF_GetFileVersion()

[Palm OS only.] Get the version number of the currently matching file.

Parameters *None*

Return Value The function returns the file version number.

Comments FF_GetFileVersion requires the FindFiles extension. This function operates on the current file found by FF_FindFirstFile or FF_FindNextFile. This function is for the Palm OS platform only, and always returns zero on the Pocket PC platform. This is NOT the application-defined version number, this is a Palm OS database-specific version number, which is seldom used.

Example

```
fileversion = FF_GetFileVersion
```

See Also FF_GetFileCreator, FF_GetFileType

FF_GetLastErr

FF_GetLastErr()

Get the last error code for FindFiles file functions.

Parameters *None*

Return Value A return value of 0 indicates no error, any other value is a file error.

Comments FF_GetLastErr requires the FindFiles extension.

Example

```
result = FF_GetLastErr
```

See Also

FF_GetNextVolRef

FF_GetNextVolRef(*bFirst*)

[Palm OS only.] Get the first or next VFS volume reference, which identifies a VFS memory card volume for use in other functions.

Parameters *bFirst* Boolean value indicating (true) if you are starting a new volume enumeration or (false) if you are getting the next available volume reference.

Return Value A volume reference number >= 1 if a volume is found, or 0 if there are no more volumes

Comments FF_GetNextVolRef requires the FindFiles extension. This function is for the Palm OS platform only, and always returns 0 on Pocket PC. Use this function in a loop to get volrefs for all of the available memory card volumes, until the function returns a 0 indicating no more volumes. NOTE: volrefs may not be sequential, so you cannot assume they are in sequence from 1..n. Volume reference numbers are used for all other “VFS” functions in the FindFiles extension.

Example

```
'enumerate all VFS volumes and add to table for droplist
dim volref, numrecs, loop
'clear all records from the table
numrecs = tables("tblVolumes").count
for loop = 1 to numrecs
      tables("tblVolumes").removerecord(numrecs - loop)
next
'add a record for internal RAM (card 0)
tables("tblVolumes").createrecord
tables("tblVolumes").movelast
tables("tblVolumes").fields("volref") = 0
tables("tblVolumes").fields("label") = "RAM"
'find the first volume
volref = FF_GetNextVolRef( True )
while volref <> 0
      tables("tblVolumes").createrecord
      tables("tblVolumes").movelast
      tables("tblVolumes").fields("volref") = volref
      tables("tblVolumes").fields("label")=FF_GetVFSLabel(volref)
      volref = FF_GetNextVolRef( false )
wend
```

See Also FF_DeviceHasVFS, FF_FindFirstFileVFS, FF_FindNextFileVFS, FF_GetVFSLabel, FF_GetVFSVolRef, FF_SetVFSVolRef

FF_GetVFSLabel

FF_GetVFSLabel(*volref*)

[Palm OS only.] Get the label of the specified VFS volume reference.

Parameters *volref* Numeric VFS volume reference, from FF_GetNextVolRef

Return Value A string containing the volume label. If there is an error, a blank string is returned. It is also possible for a memory card to have a blank label, in which case a blank string would be returned as well.

Comments FF_GetVFSLabel requires the FindFiles extension. This function is for the Palm OS platform only, and is for memory cards only.

Example *See the example for FF_GetNextVolRef*

See Also FF_DeviceHasVFS, FF_FindFirstFileVFS, FF_FindNextFileVFS, FF_GetNextVolRef, FF_GetVFSVolRef, FF_SetVFSVolRef

FF_GetVFSVolRef

FF_GetVFSVolRef()

[Palm OS only.] Gets the currently active global VFS volume reference.

Parameters *None*

Return Value A numeric volume reference number.

Comments FF_GetVFSVolRef requires the FindFiles extension. This function is for the Palm OS platform only, and is for memory cards only. The active volume reference number is set with the FF_SetVFSVolRef function.

Example

```
volref = FF_GetVFSVolRef()
```

See Also FF_DeviceHasVFS, FF_FindFirstFileVFS, FF_FindNextFileVFS, FF_GetNextVolRef, FF_GetVFSLabel, FF_SetVFSVolRef

FF_SetCaseSensitive

FF_SetCaseSensitive(*bSensitive*)

[Palm OS only.] Set the filename matching case sensitivity true or false for a new search.

Parameters *bSensitive* Boolean True or False

Return Value None

Comments FF_SetCaseSensitive requires the FindFiles extension. This function is for the Palm OS platform only, as Pocket PC FindFiles functions are always NOT case sensitive.

Example

```
'set case sensitivity False
FF_SetCaseSensitive(false)
```

FF_SetVFSVolRef

FF_SetVFSVolRef(*volref*)

[Palm OS only.] Sets the active global VFS volume reference for use with other functions.

Parameters *volref* VFS volume reference number of the desired memory card.

Return Value None

Comments FF_SetVFSVolRef requires the FindFiles extension. This function is for the Palm OS platform only, and is for memory cards only. The active volume reference number set with this function is used for several VFS related functions, and should be set before calling those functions.

Example

```
volref = FF_GetNextVolRef(True)
if volref <> 0 then
      FF_SetVFSVolRef(volref)
else
      MsgBox("Memory card not found")
endif
```

See Also FF_DeviceHasVFS, FF_FindFirstFileVFS, FF_FindNextFileVFS, FF_GetNextVolRef, FF_GetVFSLabel, FF_GetVFSVolRef

FF_ShowPrivateVolumes

FF_ShowPrivateVolumes(*bShow*)

[Palm OS only.] Specify whether to make hidden internal private volumes visible to the FF_GetNextVolRef function.

Parameters *bShow* Boolean True or False

Return Value None

Comments FF_ShowPrivateVolumes requires the FindFiles extension. This function is for the Palm OS platform only. Palm OS devices with the NVFS memory system may have hidden internal private volumes that can be accessed through this function. The hidden internal private volumes should normally be left alone.

Example

```
'show private volumes for volume enumeration
FF_ShowPrivateVolumes(true)
```

Float

Float(*Variable*)

Converts a value to floating point.

Parameter *Variable* Integer or string value to be converted to floating-point value.

Return Value Floating-point value.

Comments Use the `Float` conversion operator to convert a number to floating point. Do this to ensure that operations are carried out as floating point. For example, when multiplying two very large numbers.

The `Float` operator stops when it encounters a character. For example, the string “123ABC” is converted to the floating-point value 123.0. The string “ABC” is converted to the floating-point value 0.0.

Example

```
'Example of floating-point conversion
'InputA, InputB, and OutputA are
'edit controls.
OutputA = Float(InputA) * InputB
```

See Also Int, * [multiply], Str

FollowCursor

Extension.FollowCursor(*str*)

Tapping a control makes it next to scan.

Parameter *Str* “On” or “Off”

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions. Default for the Bar Code Reader is “Off”.

Example

```
'Example of FollowCursor(str)
BarCode1.FollowCursor("On")
```

Font

Controls(*ControlName*).Font

Returns or sets the current font of a control.

Parameter *ControlName* The name of a control.

Return Value The current FontID of the control, an integer value from 0..7 (PalmOS) or 0..3 (PocketPC) corresponding to this table of FontIDs:

FontID	Font Description PalmOS	Font Description PocketPC
0	Normal 9	Tahoma 8
1	Bold 9	Tahoma 8 Bold
2	Normal 12	Tahoma 10
3	Symbol 9	Tahoma 10 Bold
4	Symbol 11	n/a
5	Symbol 7	n/a
6	LED	n/a
7	Bold 12	n/a

Comments Font is a property of the Control object. You can change a control's font property by setting this property to one of the specified FontID values. In that case, the function does not return a value.

Example

```
'Example of Font property
'InputA is an edit control.
'change InputA font to Bold
If InputA.Font = 0 Then
  InputA.Font = 1
EndIf
```

See Also SetPosition

For…To…Next

For *Variable* = *InitialValue* To *FinalValue*
[Step] *[Increment]*
Next *Variable*
Exit For

Perform a for loop.

Parameters

Variable	The variable that serves as a counter for the loop.
InitialValue	The value to which the counter is initialized at the beginning of the loop.
FinalValue	The value of the counter during the last execution of the loop
Increment	The value that the counter increments to during the `Next` statement of each execution of the loop.

Return Value None

Comments Each `For...Next` loop begins with a *Variable* set to *InitialValue*. Typically, a loop performs some set of operations before the `Next` statement. The `Next` statement increments the value of *Variable* by the value of `Step` (*Increment)*. *Increment* can be a positive or negative value. If you omit *Increment*, it defaults to 1. If you omit *Increment,* you must also omit the keyword `Step`.

The loop continues to execute until *Variable* reaches or passes *FinalValue*. The final execution of the loop takes place with *Variable* <= *FinalValue.* The loop exit at the `Next` statement. To exit a `For` loop prematurely, use the `Exit For` statement.

You can nest multiple `For` loops. Use the `Exit For` statement to exit the current `For` loop.

Example

```
'Example of For loop
Dim x
Dim y
For x = 1 to 10
    MsgBox("The value of x is " & x)
    For y = 1 to 3
        MsgBox("The value of y is " & y)
        Next y
    If x = 4 Then Exit For
    Next x
MsgBox("Done.")
```

See Also While … Wend

FormatDate

FormatDate(*Date, FormatString*)

Formats a date string according to the format expression.

Parameters	*Date*	The date to be formatted.
	FormatString	A string describing the way in which the date string should be formatted.

Return Value String containing the date fomatted to the specified appearance.

Comments FormatDate requires the Strings extension. The Date must be in the format set in the PDA preferences. Satellite Forms always returns a date in that format, so you can plug in the data from your tables or from SysDateToDate without any modification. The FormatString determines how the resulting date string appears, and can include these values:

`yy`	2 digit year
`yyyy`	4-digit year
`m`	1-digit or 2-digit month
`mm`	2-digit month
`mmm`	3-character month
`mmmm`	Full month name
`d`	1-digit or 2-digit day
`dd`	2-digit day
`ddd`	3-character day
`dddd`	Full day name
`w`	Day of week as number
`q`	Quarter
`y`	Day of year

Example

```
'Example of the FormatDate function
dim strTest
strTest = FormatDate("4/12/05", "YYYYMMDD")
'Result: strTest contains "20050412"
```

See Also FormatDateN, FormatTime, FormatTimeN

FormatDateN

FormatDateN(*Number, FormatString*)

Formats a date string according to the format expression.

Parameters	*Number*	The date to be formatted, in system date format as returned by GetSysDate or DateToSysDate.
	FormatString	A string describing the way in which the date string should be formatted.

Return Value String containing the date fomatted to the specified appearance.

Comments FormatDate requires the Strings extension. The Number is the date expressed in system date format (days since 00:00 January 1, 1904), as returned by GetSysDate or DateToSysDate. See FormatDate for formatting details.

Example

```
'Example of the FormatDateN function
dim strTest
strTest = FormatDate(37538, "YYYY-MM-DD")
'Result: strTest contains "2006-10-10"
```

See Also FormatDate, FormatTime, FormatTimeN

FormatNumber

FormatNumber(*Number, NumDecimals*)

Returns a string representation of a number formatted with the specified number of decimal places.

Parameters	*Number*	The number to be formatted.
	NumDecimals	The desired number of decimal places.

Return Value String containing the number at the specified precision.

Comments `FormatNumber` is a method of the `App` object.

Example

```
'Example of the FormatNumber method
'InputA and OutputA are edit controls.
OutputA = FormatNumber(InputA, 2)
```

FormatTime

FormatTime(*Time, FormatString*)

Formats a time string according to the format expression.

Parameters	*Time*	The time to be formatted.
	FormatString	A string describing the way in which the time string should be formatted.

Return Value String containing the time fomatted to the specified appearance.

Comments FormatTime requires the Strings extension. The Time must be in the format set in the PDA preferences. Satellite Forms always returns a time in that format, so you can plug in the data from your tables or from SysTimeToTime without any modification. The FormatString determines how the resulting time string appears, and can include these values:

`h`	1-digit or 2-digit hour
`hh`	2-digit hour
`n`	1-digit or 2-digit minute
`nn`	2-digit minute
`m`	1-digit or 2-digit minute
`mm`	2-digit minute
`s`	1-digit or 2-digit second
`ss`	2-digit second
`AM/PM`	AM or PM
`am/pm`	am or pm
`A/P`	A or P
`a/p`	a or p

Example

```
'Example of the FormatTime function
dim strTest
strTest = FormatTime("6:32:05 PM", "HH:MM:SS")
'Result: strTest contains "18:32:05"
```

See Also FormatTimeN, FormatDate, FormatDateN

FormatTimeN

FormatTimeN(*Number, FormatString*)

Formats a time value according to the format expression.

Parameters *Number* The time to be formatted.

FormatString A string describing the way in which the time string should be formatted.

Return Value String containing the time fomatted to the specified appearance.

Comments FormatTime requires the Strings extension. The Number must be the time in the system time format (number of seconds since midnight) as returned by GetSysTime or TimeToSysTime. See FormatTime for formatting details.

Example

```
'Example of the FormatTimeN function
dim strTest
strTest = FormatTimeN(125, "HH:MM:SS")
'Result: strTest contains "00:02:05"
```

See Also FormatTime, FormatDate, FormatDateN

FREXPFRAC

FREXPFRAC(*x*)

Breaks given value into normalized fraction and an integral power of 2. Returns the fractional value.

Parameter *x* The number to raise to the integral power of 2.

Return Value The fractional value resulting from the equation.

Comments Requires the Math extension.

Example

```
'Example of FREXPFRAC(x)
Dim z
Dim x
x = 4.67
z = FREXPFRAC(x)
```

FREXPINT

FREXPINT(*x*)

Breaks given value into normalized fraction and an integral power of 2. Returns the integer value.

Parameter *x* The number to raise to the integral power of 2.

Return Value The integer value resulting from the equation.

Comments Requires the Math extension.

Example

```
'Example of FREXPINT(x)
Dim z
Dim x
x = 4.67
z = FREXPINT(x)
```

Function

Function *name* [(*arglist*)]

Defines a global script function that can take optional parameters, perform your user-defined statements, and returns a result. Global functions are available to all forms and scripts in your application.

Parameters *optional arguments*

Return Value Value defined by the function.

Comments Applies to the entire application. In the Global script section of the application property explorer window, there are two Global Funcs & Subs sections, labeled (shared) and (private). The (shared)section allows you to write global scripts that are shared between all platform targets in the application, for example a Palm target and a Pocket PC 2003 target.The (private) section allows you to write global scripts that exist in the current target only, so that you may have script that apply to the current target platform only. This enables you to have common code between targets in the (shared) section, and platform-specific code in the (private) section. One common use for this capability is to set a global variable to a specific value that indicates which platform your application is running on, thus enabling formlevel scripts to take appropriate action based on the current platform target.

Example

```
Function name [(arglist)]
[statements]
[name = expression]
[exit]
[statements]
[name = expression]
End Function
```

Remarks

- You can define local variables using the Dim keyword. The value of local variables in a Function is not preserved between calls to the routine.
- You cannot define a nested subroutine within a Sub or a Function.
- You can call a subroutine using its name and its variable. Unlike Visual Basic, you do not need to use the keyword Call.
- Unlike Visual Basic, the Exit keyword cannot be fully qualified. In Visual Basic, you have to use Exit Sub to exit the routine.
- Functions and subroutines can call themselves repeatedly (recursive). Do so cautiously because excessive recursion can lead to stack overflow.

See Also Sub

GetAppCreator

GetAppCreator

Returns a string containing the application's 4-character creatorID.

Parameter None

Return Value The application's 4-character creatorID string.

Comments `GetAppCreator` is a method of the `App` object. This is useful for any instance in which you need to know the application's 4-character creatorID string. This may be useful if your app needs to contruct a path to a file, in which the creatorID is a component of the filename (for example a table PDB file belonging to the app).

Example

```
'Example of GetAppCreator output
'SMS5
```

See Also GetAppName, GetAppPath, GetAppVersion

GetAppName

GetAppName

Returns a string containing the name of the application as defined in the project properties.

Parameter None

Return Value The application name string.

Comments `GetAppName` is a method of the `App` object. This is useful for any instance in which you need to know the application name, for example in displaying app name and version information to the user.

Example

```
'Example of GetAppName output
'My Super App
```

See Also GetAppCreator, GetAppPath, GetAppVersion

GetAppPath

GetAppPath

Returns a string containing the folder path in which the application resides on the PocketPC device.

Parameter None

Return Value The folder path in which the application resides on the PocketPC device.

Comments `GetAppPath` is a method of the `App` object. This is useful for any instance in which you need to know the path to a file, for example the path to an image file used for the ShowImage control, or a BMP file created by the InkHelper extension, or another application for the SysUtils SU_LaunchApp function, etc. The use of the GetAppPath function to get the application's folder path enables your code that relies on files paths to keep working without modification even if your application is moved to a different folder (for example to an external memory card folder). On the PalmOS platform, this function returns a blank string, because the PalmOS does not use a folder-based (hierarchical) file system.

Example

```
'Example of GetAppPath output
'\My Documents\MyApp\
```

See Also GetAppName, GetAppCreator, GetAppVersion

GetAppVersion

GetAppVersion

Returns a string containing the application major and minor version numbers.

Parameter None

Return Value The application version number as a string in the format *MMmm*.

Comments `GetAppVersion` is a method of the `App` object. This is useful for any instance in which you need to know the major and minor version numbers of your application, as defined in the project properties. This may be useful if your app needs to contruct a path to a file that includes the version number as a component of the filename (for example a table PDB file belonging to the app).

Example

```
'Example of GetAppVersion output
'0201
```

See Also GetAppName, GetAppPath, GetAppCreator

GetColor

GetColor(*ForeOrBack*)

Returns the colors currently in use.

Parameter *ForeOrBack* Set to 0 to retrieve the foreground color; set to 1 to retrieve the background color; set to 2 to retrieve the text color.

Return Value The 8-bit color value for the specified color.

Comments Applies only to the Color Graphics extension.

Example

```
'Example of GetColor(ForeOrBack)
Dim x
x = GetColor(0)
```

GetCurrentColor

GetCurrentColor(*Color*)

Specifies how often the `OnClick` event fires.

Parameter	*Color*	1 = Return foreground color.
		2 = Return background color.

Return Value The specified color value.

Comments Applies only to the Color Slider control.

Example

```
'Example of GetCurrentColor(Colors)
Dim x
x = GetCurrentColor(1)
```

GetCurrentRGBColors

GetCurrentRGBColors(*ForeOrBack*)

Returns a text string containing the current red, green, and blue colors.

Parameter	*ForeOrBack*	Set to 0 to retrieve the foreground color; set to 1 to retrieve the background color; set to 2 to retrieve the text color.

Return Value The RGB color value for the specified color.

Comments Applies only to the Color Graphics extension.

Example

```
'Example of GetCurrentRGBColors(ForeOrBack)
Dim sRGB
sRGB = GetCurrentRGBColors(0)
```

GetEngineVersion

GetEngineVersion

Returns the version number of the runtime engine as a string using *MM.mm.ii.bbb* format. The *MM* in this format represents the major version number, the *mm* represents minor version, *ii* represents the internal version, and the *bbb* represents the build number.

Parameter None

Return Value The current runtime engine version number string.

Comments `GetEngineVersion` is a method of the `App` object.

Example

```
'Example of GetEngineVersion output
'6.1.0.010
```

GetFillColor

GetFillColor()

Returns the Fill color.

Parameters None

Return Value The current Fill color value.

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of GetFillColor()
Dim x
x = GetFillColor()
```

GetFocus

Forms(*FormName*).GetFocus

Returns the index of the control that has the focus using *%Fnnn.Cnnn* format or “” if no control has focus. The *F* in this format represents form, the *C* represents control, and the *nnn* represents the index.

Parameter *FormName* Name of a form.

Return Value None

Comments `GetFocus` is a method of the `Form` object.

GetIndex

Extension.GetIndex()

Returns the index of the current control.

Parameters None

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of GetIndex()
Dim x
x = BarCode1.GetIndex()
```

GetLastKey

GetLastKey(*ASCII_Char, VirtualKeyCode, Modifiers*)

Returns the last key the user pressed.

Parameters

ASCII_Char ASCII code of the last character entere, or zero if the key pressed was a virtual key code, such as the Find key.

VirtualKeyCode Virtual key code of the last character entered.

Modifiers Modifiers of the last character entered.

Return Value None

Comments `GetLastKey` is a method of the `App` object. The last key entered can be one of the plastic buttons, one of the silk-screened buttons, or a Graffiti stroke. Call this function from within the `OnKey` event handler to intercept a user’s keystrokes or button presses. To consume the key, use `Fail` to exit from `OnKey`. To allow the key to be processed by the handheld device OS, use `Exit`.

For a list of ASCII and Virtual Key codes, see the Palm OS SDK documentation.

Modifiers can take the following values:

Shift Key	& H01
Caps Lock	&H02
Num Lock	&H04
Command Key	&H08
Auto-Repeat	&H40
Power On	&H100

GetPenColor

GetPenColor()

Returns the Pen color.

Parameters None

Return Value The current Pen color value.

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of GetPenColor()
Dim x
x = GetPenColor()
```

GetPenStatus

GetPenStatus(*x,y*)

Indicates whether the stylus is touching the screen and the x, y coordinates if it is.

Parameters

X	On return, contains the *x* coordinate of stylus position.
Y	On return, contains the *y* coordinate of the stylus position.

Return Value TRUE if the stylus is touching the screen; FALSE if it is not.

Comments `GetPenStatus` is a method of the `App` object. If it returns TRUE, the *x* and *y* coordinates are valid; otherwise, they are undefined.

GetPosition

Controls(*ControlName*).GetPosition(cX, cY, cW, cH)

Returns the current position (cX, cY) and size (cW, cH) of a control.

Parameters

ControlName	Name of a control.
cX	On return, contains the top left X coordinate of the control.
cY	On return, contains the top left Y coordinate of the control.
cW	On return, contains the width of the control.
cH	On return, contains the height of the control.

Return Value None, the location and size values are returned in the cW, cY, cW, and cH parameter variables.

Comments GetPosition is a method of the Control object. This method is useful when combined with the new Dynamic Input Area support for PalmOS that enables you to move and resize controls on a form in response to changes in the form size or orientation.

Example

```
'example of control GetPosition and SetPosition methods
Dim cX, cY, cW, cH
'obtain the current location and size of Button1
Button1.GetPosition(cX, cY, cW, cH)
'move Button1 control down 10 pixels, right 10 pixels
'and widen by 5 pixels, increase height by 5 pixels
Button1.SetPosition(cX+10, cY+10, cW+5, cH+5)
```

See Also SetPosition

GetRecordAdvMode

Extension.GetRecordAdvMode()

Returns the Record Advance Mode setting.

Parameters None

Return Value The current Record Advance Mode setting:

0=Off

1=Always Create (Bar Code Reader); On (Symbol Integrated Scanner)

2=Create At End (Bar Code Reader); Always Create (Symbol Integrated Scanner)

3=On (Bar Code Reader); Create At End (Symbol Integrated Scanner)

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of GetRecordAdvMode()
Dim x
x = BarCode1.GetRecordAdvMode()
```

GetScan

Extension.GetScan(*timeout*)

Returns a string of scanned data.

Parameter *timeout* Set to 0 to clear the scan buffer. Must be set to 0 for Symbol Integrated Scanner.

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions. You must call `EnableScanner` before using this method with the Symbol Integrated Scanner, otherwise a fatal exception may occur.

Example

```
'Example of GetScan(timeout)
BarCode1.GetScan(0)
```

See Also EnableScanner

GetScreenHeight

GetScreenHeight()

[Pocket PC only.] Return screen height in pixels.

Parameter *None*

Return Value Screen height in pixels.

Comments Requires the ScreenSize extension for Pocket PC.

Example

```
'Example of GetScreenHeight
MsgBox("Screen height is " &GetScreenHeight &" pixels.")
```

See Also

GetScreenSize

GetScreenSize()

[Pocket PC only.] Return screen dimensions in string WWWxHHH.

Parameter *None*

Return Value Screen dimensions (pixels) in string WWWxHHH

Comments Requires the ScreenSize extension for Pocket PC.

Example

```
'Example of GetScreenSize
MsgBox("Screen dimensions are " &GetScreenSize)
```

See Also

GetScreenWidth

GetScreenWidth()

[Pocket PC only.] Return screen width in pixels.

Parameter *None*

Return Value Screen width in pixels.

Comments Requires the ScreenSize extension for Pocket PC.

Example

```
'Example of GetScreenWidth
MsgBox("Screen width is " &GetScreenWidth &" pixels.")
```

See Also

GetSelection

Controls(*ControlName*).GetSelection(*StartSel*, *EndSel*)

The `GetSelection` method is available only for Paragraph and Edit controls. It retrieves the start and end offsets of the highlighted text in the control.

Parameters *ControlName* Name of control.

StartSel Offset to beginning of selection.

EndSel Offset to end of selection.

Return Value None

Comments `GetSelection` is a method of the `Control` object (Paragraph and Edit controls only). The selection is returned in the parameters *StartSel* and *EndSel*, both of which must be variables. *StartSel* contains the offset of the beginning of the selection. The offset of the first character in a control is 0. *EndSel* contains the offset to the character following the end of the selection. For example, if a control contains a string “ABCD” and the user highlights the “BC” portion, this method returns *StartSel* = 1 and *EndSel* = 3. If *StartSel* is the same as *EndSel*, no text is highlighted.

GetSysDate

GetSysDate

Returns the number of days since January 1, 1904.

Parameters None

Return Value Days since January 1, 1904

Comments `GetSysDate` is a method of the `App` object.

Example

```
'Example of the GetSysDate method
'OutputA and Output B are edit controls.
'Sample output is 38899.
OutputA = GetSysDate
'Sample output is 07/02/10.
OutputB = SysDateToDate(GetSysDate)
```

See Also GetSysTime, SysDateToDate, DateToSysDate

GetSysTime

GetSysTime

Returns the number of seconds since 12:00 AM on January 1, 1904.

Parameters None

Return Value Seconds since 12 a.m. January 1, 1904

Comments `GetSysTime` is a method of the `App` object.

Example

```
'Example of the GetSysTime method
'OutputA and Output B are edit controls.
'Sample output is -1327064373.
OutputA = GetSysTime
'Sample output is 10:36:20 pm.
OutputB = SysTimeToTime(GetSysTime)
```

See Also GetSysDate, SysTimeToTime, TimeToSysTime

GetTickCount

GetTickCount

Returns the number of timer ticks since the handheld device was last reset. Ticks do not advance when the device is off.

Parameters None

Return Value Number of timer ticks since the handheld was last reset.

Comments `GetTickCount` is a method of the `App` object.

See Also KillTimer, SetTimer, GetTickFrequency

GetTickFrequency

GetTickFrequency

Returns the frequency of the timer ticks.

Parameters None

Return Value Number of ticks per second.

Comments `GetTickFrequency` is a method of the `App` object.

See Also KillTimer, SetTimer, GetTickCount

GetUserID

GetUserID

Returns the Satellite Forms-assigned unique user ID of the handheld device.

Parameters None

Return Value Satellite Forms-assigned unique user ID of the handheld device.

Comments `GetUserID` is a method of the `App` object.

Example

```
'Example of GetUserID method
'OutputA is an edit control.
OutputA = GetUserID
```

GetUserName

GetUserName

Returns the Satellite Forms-assigned unique user name of the handheld device.

Parameters None

Return Value Satellite Forms-assigned unique user name of the handheld device.

Comments `GetUserName` is a method of the `App` object.

Example

```
'Example of GetUserName method
'OutputA is an edit control.
OutputA = GetUserName
```

GPS_CalcDistance

GPS_CalcDistance(*Lat1, Lon1, Lat2, Lon2*)

Calculate the distance in metres between two GPS waypoints.

Parameters *Lat1* Latitude in decimal degrees of first waypoint.
Lon1 Longitude in decimal degrees of first waypoint.
Lat2 Latitude in decimal degrees of second waypoint.
Lon2 Longitude in decimal degrees of second waypoint.

Return Value Returns the distance in metres between two waypoints.

Comments GPS_CalcDistance requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions. Latitudes and longitudes must be supplied in decimal degrees, which is the same format as returned by the GPS_GetPosLatitude and GPS_GetPosLongitude functions. Positive latitude indicates N, and negative latitude indicates S. Positive longitude indicates E, and negative longitude indicates W.

Example

```
'Example of GPS_CalcDistance method
'Canyon campground in Yellowstone National Park
edLat1 = 44.735300
edLon1 = -110.488083
'Bridge Bay campground in Yellowstone National Park
edLat2 = 44.534500
edLon2 = -110.436967
'Distance in metres between the two campgrounds
edDistance = GPS_CalcDistance(edLat1, edLon1, edLat2, edLon2)
```

See Also GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_HasGPSAPI, GPS_OpenGPS

GPS_CloseGPS

GPS_CloseGPS()

Close connection to GPS receiver, powering down GPS receiver if no other tasks are also using it.

Parameters *None*

Return Value Returns 0 if no error, or error code if GPS conection could not be opened.

Comments GPS_CloseGPS requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions. Call GPS_CloseGPS when you are done using the GPS in your application, to conserve power.

See Also GPS_CalcDistance, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_HasGPSAPI, GPS_OpenGPS

GPS_GetPosLatitude

GPS_GetPosLatitude(*maxdataage*)

Get Latitude in decimal degrees from GPS position data.

Parameters *maxdataage* Maximum age in milliseconds of GPS data that you will accept as current. If the data is older than that, it will be considered invalid.

Return Value Returns 0 if no valid data, -1 if the GPS connection has not been opened, or latitude data in decimal degrees. Positive numbers indicate North latitude, negative numbers indicate South latitude.

Comments GPS_GetPosLatitude requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions.

Example

```
'Example of GPS_GetPosLatitude function
edLat = GPS_GetPosLatitude(edMaxDataAge)
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_HasGPSAPI, GPS_OpenGPS

GPS_GetPosLongitude

GPS_GetPosLongitude(*maxdataage*)

Get Longitude in decimal degrees from GPS position data.

Parameters *maxdataage* Maximum age in milliseconds of GPS data that you will accept as current. If the data is older than that, it will be considered invalid.

Return Value Returns 0 if no valid data, -1 if the GPS connection has not been opened, or longitude data in decimal degrees. Positive numbers indicate East longitude, negative numbers indicate West longitude.

Comments GPS_GetPosLongitude requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions.

Example

```
'Example of GPS_GetPosLongitude function
edLon = GPS_GetPosLongitude(edMaxDataAge)
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_HasGPSAPI, GPS_OpenGPS

GPS_GetPosOther

GPS_GetPosOther(*maxdataage*, *otherdatatype*)

Get other GPS position data, indicated by other information type.

Parameters *maxdataage* Maximum age in milliseconds of GPS data that you will accept as current. If the data is older than that, it will be considered invalid.

otherdatatype Index of other data type to obtain (see Comments for list).

Return Value Returns 0 if no valid data, -1 if the GPS connection has not been opened, or requested data value.

Comments GPS_GetPosLatitude requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions. The list of other data types and their index numbers is shown below.

1	UTC_TIME
2	LATITUDE
4	LONGITUDE
8	SPEED
16	HEADING
32	MAGNETIC_VARIATION
64	ALTITUDE_WRT_SEA_LEVEL
128	ALTITUDE_WRT_ELLIPSOID
256	POSITION_DILUTION_OF_PRECISION
512	HORIZONTAL_DILUTION_OF_PRECISION
1024	VERTICAL_DILUTION_OF_PRECISION
2048	SATELLITE_COUNT
4096	SATELLITES_USED_PRNS
8192	SATELLITES_IN_VIEW
16384	SATELLITES_IN_VIEW_PRNS
32768	SATELLITES_IN_VIEW_ELEVATION
65536	SATELLITES_IN_VIEW_AZIMUTH
131072	SATELLITES_IN_VIEW_SIGNAL_TO_NOISE_RATIO
1048576	GPS_FIX_QUALITY

	2097152	GPS_FIX_TYPE
	4194304	GPS_FIX_SELECTION

Example

```
'Example of GPS_GetPosOther function to get current speed
speed = GPS_GetPosOther(8)
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_HasGPSAPI, GPS_OpenGPS

GPS_GetPosUTCTime

GPS_GetPosUTCTime(*maxdataage*)

Get UTC Time from GPS position data.

Parameters *maxdataage* Maximum age in milliseconds of GPS data that you will accept as current. If the data is older than that, it will be considered invalid.

Return Value Returns 0 if no valid data, -1 if the GPS connection has not been opened, or or UTC Time value in system time format.

Comments GPS_GetPosUTCTime requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions. UTCtime is in SatForms system time format which is seconds since 00:00 Jan 1, 1904. That value can be converted into both a date and a time, as shown in the example below.

Example

```
'Example of GPS_GetUTCTime function
'we can retrieve both the date and time from that value
'this value is a signed 32 bit integer and can be negative
'so for date we convert to a positive by adding 2 ^ 32 to
'the returned value and converting that from seconds into
'days (divide secs by 86400 secs/day)
dim UTCtime, UInt32MAX
'2 ^ 32 = max size of 32 bit unsigned integer
UInt32MAX = 4294967296
UTCtime = GPS_GetPosUTCTime(edMaxDataAge)
edStatus = "GetPosUTCTime = " &UTCTime
if (UTCTime <> 0) and (UTCTime <> -1) then
      'convert to user readable date and time
      edDate = SysDateToDate((UTCTime + UInt32MAX) / 86400)
      edTime = SysTimeToTime(UTCTime)
else
      edDate = 0
      edTime = 0
endif
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetValidFields, GPS_HasGPSAPI, GPS_OpenGPS

GPS_GetValidFields

GPS_GetValidFields(*maxdataage*)

Check whether GPS is returning valid position data.

Parameters *maxdataage* Maximum age in milliseconds of GPS data that you will accept as current. If the data is older than that, it will be considered invalid.

Return Value Returns 0 if no valid position data fields, -1 if the GPS connection has not been opened, or positive value indicating which fields have valid data (each field represented by a bitfield value). The bitfield values are the same as those listed in the GPS_GetPosOther reference.

Comments GPS_GetValidFields requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions.

Example

```
'Example of GPS_GetValidFields function
'Check if Speed value is valid
dim k_Speed, valid
k_Speed = 8
valid = GPS_GetValidField(edMaxDataAge)
if valid and k_Speed = true then
      MsgBox("Speed data is valid!")
endif
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_HasGPSAPI, GPS_OpenGPS

GPS_HasGPSAPI

GPS_HasGPSAPI()

Check whether the current device has the Windows Mobile GPS API.

Parameters *None*

Return Value Returns True if device has GPS API, or False if not. Windows Mobile version 5 and higher devices should have the GPS API, but older versions and Windows CE devices do not.

Comments GPS_HasGPSAPI requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions.

Example

```
'Example of GPS_HasGPSAPI function
if not GPS_HasGPSAPI then
      MsgBox("Device does not have GPS API")
endif
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_OpenGPS

GPS_OpenGPS

GPS_OpenGPS()

Opens connection to GPS receiver, powering up GPS receiver if necessary.

Parameters *None*

Return Value Returns 0 if no error, or error code if GPS conection could not be opened.

Comments GPS_OpenGPS requires the GPS extension. Use GPS_HasGPSAPI function to determine if current device supports GPS functions. Call GPS_OpenGPS when you start using the GPS in your application, before calling other functions to retrieve data fields. Remember to call GPS_CloseGPS when done using it, to conserve power.

Example

```
'Example of GPS_OpenGPS function
dim result
if GPS_HasGPSAPI then
      result = GPS_OpenGPS
      if result = 0 then
            g_GPSOpen = true
            MsgBox("GPS Opened!")
      endif
else
      MsgBox("Device does not have GPS")
endif
```

See Also GPS_CalcDistance, GPS_CloseGPS, GPS_GetPosLatitude, GPS_GetPosLongitude, GPS_GetPosOther, GPS_GetPosUTCTime, GPS_GetValidFields, GPS_HasGPSAPI

HexStringFromInt

HexStringFromInt(*value*)

Converts an integer value to a hexadecimal string.

Parameter	*Value*	Integer value to be converted to a hexadecimal string.

Return Value Hexadecimal string.

Comments HexStringFromInt requires the Strings extension.

Example

```
'Example of HexStringFromInt
Dim hexvalstr
hexvalstr = HexStringFromInt(54321)
'Result: hexvalstr contains "D431"
```

See Also Int, IntFromHexString, Float, Str, Bool

If … Then

If *Condition1* Then
Statement1
[ElseIf] *[Condition2]* [Then]
[Statement2]
[Else]
[Statement3]
EndIf

If *Condition* Then *Statement1 : [Statement2]*

Execute different statements conditionally.

Parameters	*Condition1*	First condition to be evaluated.
	Statement1	Commands to execute if *Condition1* evaluates to TRUE.
	Condition2	Condition to evaluate if *Condition1* evaluates to FALSE.
	Statement2	Commands to execute if *Condition2* evaluates to TRUE.
	Statement3	Commands to execute if *Conditions1 and* 2 evaluate to FALSE.

Return Value None

Comments There are two ways to construct `If ...Then` statements: the block format and the single-line format.

The first usage example above illustrates the block format. In this example, if *Condition1* evaluates to TRUE, *Statement1* executes and the If statement exits without evaluating *Condition2*. If *Condition1* evaluates to FALSE, *Condition2* is evaluated. If *Condition2* evaluates to TRUE, *Statement2* executes and the If statement exits. If *Condition2* evaluates to FALSE, *Statement3* executes and the If statement exits.

Using the block format, you can nest multiple `If` statements. The `ElseIf` and `Else` clauses are optional. Each `If` statement can have multiple `ElseIf` clauses, but only a single `Else` clause.

The second usage example above illustrates the simplified single-line format. In this example, if *Condition* evaluates to TRUE, the `Then` clause executes. If *Condition* evaluates to FALSE, the `Then` clause does not execute. Multiple statements can be included in the `Then` clause by separating each statement with a colon.

Example

```
'Example of using the If statement
'InputA and InputB are edit controls.
If Float(InputA) > Float(InputB) Then
    MsgBox("The greater number is " & InputA)
ElseIf Float(InputB) > Float(InputA) Then
    MsgBox("The greater number is " & InputB)
Else
    MsgBox("The two numbers are equal")
EndIf
```

ILOGB

ILOGB(*x*)

Binary exponent of non-zero *x*. Returns an integer.

Parameter *x* The number to raise to the binary exponent.

Return Value The integer value resulting from the equation.

Comments Requires the Math extension.

Example

```
'Example of ILOGB(x)
Dim z
Dim x
x = 4
z = ILOGB(x)
```

IH_BMPColorSettings

IH_BMPColorSettings(*fgRed, fgGreen, fgBlue, bgRed, bgGreen, bgBlue*)

Set foreground and background colors for monochrome BMP file.

Parameters

fgRed	red component of RGB value for foreground color
fgGreen	green component of RGB value for foreground color
fgBlue	blue component of RGB value for foreground color
bgRed	red component of RGB value for background color
bgGreen	green component of RGB value for background color
bgBlue	blue component of RGB value for background color

Return Value None

Comments Requires the InkHelper extension. This method enables you to specify desired foreground & background colors for the BMP file created with IH_InkFieldToBitmap function. Its primary use is to allow you to invert BMP colors to work around a bitmap printing bug with PrintBoy on Windows Mobile 5 devices. The default color settings if none are specified are: fgRed = 0, fgGreen = 0, fgBlue = 0 [foreground = black], bgRed = 255, bgGreen = 255, bgBlue = 255 [background = white].

Example

```
'Example of IH_BMPColorSettings method
'Invert colors for printing with PrintBoy on WM5 devices
'use fgRed = 255, fgGreen = 255, fgBlue = 255 (white)
'and bgRed = 0, bgGreen = 0, bgBlue = 0 (black)
IH_BMPColorSettings( 255, 255, 255, 0, 0, 0)
```

See Also IH_InkFieldToBitmap

IH_DeleteFile

IH_DeleteFile(*Filename*)

Delete the specified file.

Parameters *Filename* Name of file to delete.

Return Value If the file was deleted, returns 0, else returns 1 to indicate an error.

Comments Requires the InkHelper extension. For Palm OS internal memory files, do not use any path, just the case sensitive filename. For Pocket PC files and Palm OS external memory files, specify the full path and name of the file to be deleted. This function is provided for deleting BMP files created with IH_InkFieldToBitmap, but can be used to delete any file that is not currently open and in use.

Example

```
'Example of IH_DeleteFile function
Dim result, filename, filepath
filename = “SIGNATURE1.BMP”
If g_Platform = “PALMOS” then
    filepath = “”
Else
    filepath = “\My Documents\My App\”
EndIf
result = IH_DeleteFile(filepath & filename)
```

See Also IH_InkFieldToBitmap

IH_FileToBinField

IH_FileToBinField(*Filename,* Tables(*TableName*).Fields(*InkFieldName*).Index, *row*)

Import a file into a binary field.

Parameters

Filename	Path and name of file to import.
TableName	Name of table.
InkFieldName	Name of ink field.
row	Row number of table record.

Return Value If the file was imported, returns 0, else returns 1 to indicate an error.

Comments Requires the InkHelper extension. For Palm OS internal memory files, do not use any path, just the case sensitive filename. For Pocket PC files and Palm OS external memory files, specify the full path and name of the file to be imported. It is intended for use with BMP files but can be used for any file (for PalmOS internal memory files, only streamed databases such as BMP files created with IH_InkFieldToBitmap are supported, not regular PRC/PDB databases). This is useful when you want to access a file created on the handheld in your server database, with the file stored in a table binary field. The binary field specified should NOT be the ink binary field.

Example

```
'Example of IH_FileToBinField function
dim row, strFilename, result
strFilename = GetAppPath & edName
result = IH_FileToBinField( strFilename,
Tables("tInkdata").Fields("Binfile").Index, Forms().CurrentRecord)
If (result <> 0) Then
    MsgBox("IH_FileToBinField error:" &result)
EndIf
```

See Also IH_InkFieldToBitmap, IH_InkFieldToHexText, IH_FileToHexText

IH_FileToHexText

IH_FileToHexText(*Filename*)

Return the contents of a file as HexText.

Parameters *Filename* Name of file to convert into HexText.

Return Value String containing the HexText representation of the file.

Comments Requires the InkHelper extension. For Palm OS internal memory files, do not use any path, just the case sensitive filename. For Pocket PC files and Palm OS external memory files, specify the full path and name of the file to be converted. It is intended for use with BMP files but can be used for any file (for PalmOS internal memory files, only streamed databases such as BMP files created with IH_InkFieldToBitmap are supported, not regular PRC/PDB databases). HexText is an ASCII text representation of binary data, in which the hexadecimal value of each binary byte is converted to 2 ASCII characters that indicate the binary hex value. For example, the single binary byte 0xA5 is represented in HexText as the ASCII string “A5”. When the binary data is converted into HexText, it can then be transported via methods that only support plain text (for example using TCPIP sockets via the Internet or Winsock extensions, or POSTing to a web server using HTTP). The HexText can then be converted back into binary form on the destination/host system, for example converting the HexText back into a standard BMP file.

Example

```
'Example of IH_FileToHexText function
Dim hextext, filename, filepath
filename = “SIGNATURE1.BMP”
If g_Platform = “PALMOS” then
    filepath = “”
Else
    filepath = “\My Documents\My App\”
EndIf
hextext = IH_FileToHexText(filepath & filename)
```

See Also IH_InkFieldToBitmap, IH_InkFieldToHexText, IH_FileToUUEText

IH_FileToUUEText

IH_FileToUUEText(*Filename*)

Return the contents of a file as uuencoded ASCII text.

Parameters *Filename* Name of file to convert into uuencoded ASCII text.

Return Value String containing the uuencoded text representation of the file.

Comments Requires the InkHelper extension. For Palm OS internal memory files, do not use any path, just the case sensitive filename. For Pocket PC files and Palm OS external memory files, specify the full path and name of the file to be converted. It is intended for use with BMP files but can be used for any file (for PalmOS internal memory files, only streamed databases such as BMP files created with IH_InkFieldToBitmap are supported, not regular PRC/PDB databases). Uuencoded text is a 7-bit ASCII text representation of 8-bit binary data, in standard use for many years on the Internet. When the binary data is converted into uuencoded text, it can then be transported via methods that only support plain text (for example using TCPIP sockets via the Internet or Winsock extensions, or POSTing to a web server using HTTP). The uuencoded text can then be converted back into binary form on the destination/host system, for example converting the text back into a standard BMP file.

Example

```
'Example of IH_FileToUUEText function
Dim uuetext, filename, filepath
filename = "SIGNATURE1.BMP"
If g_Platform = "PALMOS" then
    filepath = ""
Else
    filepath = "\My Documents\My App\"
EndIf
uuetext = IH_FileToUUEText(filepath & filename)
```

See Also IH_InkFieldToBitmap, IH_InkFieldToHexText, IH_FileToHexText

IH_InkFieldToBitmap

IH_InkFieldToBitmap(Tables(*TableName*).Fields(*InkFieldName*).Index, *row*, *BMPfilename*)

Save the contents of an ink field to a BMP file.

Parameters

TableName	Name of table.
InkFieldName	Name of ink field.
row	Row number of table record.
BMPfilename	Name of BMP file to create.

Return Value If the BMP file was created, returns 0, else returns a numeric error code.

Comments Requires the InkHelper extension. For Palm OS internal memory files, do not use any path, just the case sensitive filename. For Pocket PC files and Palm OS external memory files, specify the full path and name of the file to be deleted. This function enables you to convert ink images (eg. signatures, sketches) into the standard BMP format for easy use with other software, including printing from the handheld. The IH_InkFieldToBitmap and IH_InkFieldToHexText functions work with ink data that is saved in a table field, not with ink data that is in a form control. Thus, if you are wanting to convert ink from the current form control, you must save that ink control data to the table first. To do that, use Forms().Refresh, or simply access the table field after the form data has been saved to the table by moving to a new record, new form, new page, etc.

Example

```
'Example of IH_InkFieldToBitmap function
Dim result, filename, filepath
filename = "SIGNATURE" &Str(Forms().CurrentRow) &".BMP"
If g_Platform = "PALMOS" then
    filepath = ""
Else
    filepath = "\My Documents\My App\"
EndIf
result = IH_InkFieldToBitmap(Fields("Signature").index,
Forms().CurrentRow, filepath & filename)
```

See Also IH_BMPColorSettings, IH_DeleteFile, IH_FileToHexText, IH_FileToUUEText, IH_InkFieldToHexText

IH_InkFieldToHexText

IH_InkFieldToHexText(Tables(*TableName*).Fields(*InkFieldName*).Index, *row*)

Return the contents of an ink field as HexText.

Parameters

TableName	Name of table.
InkFieldName	Name of ink field.
row	Row number of table record.

Return Value String containing the HexText representation of the ink field binary data.

Comments Requires the InkHelper extension. For Palm OS internal memory files, do not use any path, just the case sensitive filename. For Pocket PC files and Palm OS external memory files, specify the full path and name of the file to be deleted. This function enables you to convert ink images (eg. signatures, sketches) in SatForms binary ink format into HexText. HexText is an ASCII text representation of binary data, in which the hexadecimal value of each binary byte is converted to 2 ASCII characters that indicate the binary hex value. For example, the single binary byte 0xA5 is represented in HexText as the ASCII string "A5". When the binary data is converted into HexText, it can then be transported via methods that only support plain text (for example using TCPIP sockets via the Internet or Winsock extensions, or POSTing to a web server using HTTP). The HexText can then be converted back into binary form on the destination/host system, for example converting the HexText back into SatForms binary ink format for display in the Ink View OCX. The IH_InkFieldToBitmap and IH_InkFieldToHexText functions work with ink data that is saved in a table field, not with ink data that is in a form control. Thus, if you are wanting to convert ink from the current form control, you must save that ink control data to the table first. To do that, use Forms().Refresh, or simply access the table field after the form data has been saved to the table by moving to a new record, new form, new page, etc.

Example

```
'Example of IH_InkFieldToHexText function
Dim hextext
hextext = IH_InkFieldToBitmap(Fields("Signature").index,
Forms().CurrentRow)
```

See Also IH_FileToHexText, IH_InkFieldToBitmap

IH_PalmFileSettings

IH_PalmFileSettings(*volumenumber, filetype, filecreatorid, filebackup*)

Specify PalmOS file settings used for other InkHelper functions that access files.

Parameters

volumenumber	Specify the volumenumber as: 0 (internal streamed files), or 1 (first VFS volume found), or *n* (*n*th VFS volume found), etc.
filetype	4-character case sensitive file type for streamed files
filecreatorid	4-character case sensitive Creator ID for streamed files
filebackup	True or False to set the backup bit on streamed files

Return Value None

Comments Requires the InkHelper extension. Palm OS only, not applicable on Pocket PC. Call this function on PalmOS **before** the other functions that work with files (eg. IH_InkFieldToBitmap). The Palm OS uses volume reference numbers to access external memory cards, which you can specify using the *volumenumber* parameter. Set *volumenumber* to 0 for internal memory files. The *filetype* should normally be “strm”. For compatibility with the PalmDataPro SFInkView conduit, set the *filecreatorid* to “EWIV”, otherwise use your application’s Creator ID. If the backup bit is set on BMP files (by setting *filebackup* to True), the Palm Backup conduit will back up the BMP files as streamed file PDBs in the user's Backup folder. You can convert these streamed file PDBs to standard Windows BMPs using the freeware PAR utility.

Example

```
'Example of IH_PalmFileSettings method
'using internal memory and SFInkView conduit compatibility
IH_PalmFileSettings(0,“strm”, “EWIV”, false)
```

See Also IH_InkFieldToBitmap, IH_DeleteFile, IH_FileToHexText, IH_FileToUUEText

Index

Object.Index

Returns the index of an object.

Parameters Object An object.

Return Value Index of object.

Comments `Index` is a read-only property of the `Control` object, the `Field` object, the `Form` object, the `Extension` object, and the `Table` object. Use the `Index` property to pass a reference to a control, field, form, or table to an extension. For more information, see the Slider SFX Custom control extension in the Samples\Extensions\Slider\Src directory.

InsertionSort

Tables(***TableName***).InsertionSort(***ColumnName***, ***Direction***)

Sorts the records in the specified table using the specified column as the key. You may sort in ascending or descending order.

Parameters

TableName Name of table.

ColumnName Name of a column.

Direction Sort order. Use TRUE for ascending and FALSE for descending.

Return Value None

Comments `InsertionSort` is a method of the `Table` object. Using this method preserves the relative order of the previous sort when you sort twice in a row on two different keys. For example, you can sort first by NAME and then by AGE. The result is be a table sorted by AGE and, within the same age groups, by NAME.

Example

```
'Example of InsertionSort method
'Sort table by first name.
Tables("MyTable").InsertionSort ("FirstName",TRUE)
```

See Also QuickSort

InStr

InStr(***String, Substring, StartPos***)

Finds occurrence of one string within another, starting at the specified start position.

Parameters *String* String to search within.
SubString SubString to search for in String.
StartPos Position in String to start searching at.

Return Value If the substring is not found, returns 0. If found, returns position where found.

Comments Requires the Strings extension.

Example

```
'Example of InStr function
Dim sPos
sPos = InStr("CDABCDE","CD",3)
'Result: sPos contains 5
```

Int

Int (*Variable*)

Converts a value to an integer.

Parameter *Variable* Floating-point or string value to be converted to an integer.

Return Value Integer value.

Comments Truncates decimal places during conversion.

Example

```
'Example of Integer Conversion
'InputA, InputB, OutputA, and OutputB are edit controls.
InputB = 2.54
OutputA = Int(InputA)
OutputB = Int(InputB)
```

See Also Float, Str, Bool, Int64

Int64

Int64(*Val*)

Converts a value to a 64-bit integer.

Parameter *Val* Floating-point or string value to be converted to a 64-bit integer.

Return Value 64-bit integer value.

Comments Truncates decimal places during conversion.

Example

```
'Example of Int64 Conversion
'InputA, InputB, OutputA, and OutputB are edit controls.
InputB = 2.54
OutputA = Int64(InputA)
OutputB = Int64(InputB)
```

See Also Int, Float, Str, Bool

IntFromHexString

IntFromHexString(*HexValStr*)

Converts a string of hexadecimal to an integer.

Parameter *HexValStr* Hexadecimal string value to be converted to an integer.

Return Value Integer value.

Comments IntFromHexString requires the Strings extension.

Example

```
'Example of IntFromHexString
Dim value
value = IntFromHexString(“AEBD”)
'Result: value contains 44733
```

See Also HexStringFromInt, Int, Float, Str, Bool

InvertText

InvertText(*Strg*, *X*, *Y*)

Inverts the text at the specified point.

Parameters *Strg* Text to draw.
X Y-origin of the upper left corner of the text in pixels.
Y X-origin of the upper left corner of the text in pixels.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
‘Example of Invert(Strg, X, Y)
InvertText("I’m here", 40, 40)
```

See Also DrawText

Is16BitCapable

Is16BitCapable()

Determines whether the handheld device is capable of displaying 16-bit color.

Parameters None

Return Value TRUE (non-zero) if the handheld device supports 16-bit color; FALSE (0) if it does not.

Comments Applies only to the Color Graphics extension.

Example

```
‘Example of Is16BitCapable()
Dim b16BitClr
b16BitClr = Is16BitCapable()
```

Is35

Is35()

Checks to see if the current handheld has Palm OS 3.5 or greater.

Parameters None

Return Value TRUE if the handheld has Palm OS 3.5 or greater; FALSE if not.

Comments Applies only to the Color Graphics extension.

Example

```
'Example of Is35()
Dim bIsROM35
bIsROM35 = Is35()
```

IsEmpty

IsEmpty(x)

Tests a table field or variable for an Empty state.

Parameters None

Return Value None

Comments Applies only to table fields and to variables, but not to controls.

Fields in a new record are always initialized to Empty, which is not equal to "" (empty string) or 0. New variables are also initialized to Empty. Once you assign a value to a field or variable, it is no longer Empty. However, you can make the table field or variable empty again by assigning it to Empty. You can test whether a field or variable is empty using the IsEmpty(object) keyword.

Caution: Empty and IsEmpty *must not* be used with controls. Note: AppDesigner does not currently enforce this rule when compiling your application, so you need to make sure not to use Empty/IsEmpty with controls!

Example

```
'Example of assigning Empty to a var and testing with IsEmpty
x = Empty
if IsEmpty(x) then msgbox("x is Empty!")

'Example of testing a table field with IsEmpty
if IsEmpty(Fields(fieldname)) then msgbox("Field is empty!")
```

See Also Empty

IsHandheld35

Extension.IsHandheld35()

Checks to see if the current handheld has Palm OS 3.5 or greater.

Parameters None

Return Value TRUE if the handheld has Palm OS 3.5 or greater; FALSE if not.

Comments Applies only to the Color Slider control.

Example

```
'Example of IsHandheld35()
Dim bIsHH35
bIsHH35 = ColorSlider1.IsHandheld35()
```

ISINF

ISINF(*x*)

Evaluates a number for its relationship to positive or negative infinity.

Parameter *x* The number to raise to the binary exponent.

Return Value 0 if the specified value is finite or not a number;

+1 if the specified value approaches positive Infinity;

-1 if the specified value approaches negative Infinity.

Comments Requires the Math extension.

Example

```
'Example of ISINF(x)
Dim z
z = ISINF(x)
```

KillTimer

KillTimer

Turns off the timer.

Parameters None

Comments `KillTimer` is a method of the `App` object.

See Also SetTimer

LaunchURL

LaunchURL(*strURL*)

Launch a specified URL or view local html and image files in the default device web browser.

Parameters *strURL* String containing full URL of website to launch, or local HTML or image files to view in the default device web browser.

Return Value Returns 0 if successful, or error code if not.

Comments Requires the LaunchURL extension.

Example

```
'Example of LaunchURL
dim strURL, err
strURL = "http://www.satelliteforms.net/"
err = LaunchURL(strURL)
```

LBackSp

LBackSp()

Sends a Backspace character to the printer.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LBackSp()
LBackSp()
```

LBell

LBell()

Sends a Bell character to the printer.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LBell()
LBell()
```

LCancel

LCancel()

Sends a CAN character to the printer.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LCancel()
LCancel()
```

LCase

LCase(*string*)

Converts string to all lower case.

Parameter *string* The string to convert to all lower case.

Return Value The input string converted to all lower case.

Comments Requires the Strings extension.

Example

```
'Example of LCase
Dim strTest
strTest = LCase(“STRING TO CONVERT”)
‘returns “string to convert”
```

LCondensed

LCondensed(*Enable*)

Sets the printer’s condensed print mode.

Parameter	*Enable*	TRUE enables condensed mode; FALSE disables condensed mode.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LCondensed(Enable)
LCondensed(TRUE)
```

LDDGraphics

LDDGraphics(*n1*, *n2*)

Sets double density graphics mode for the specified number of following bytes.

Parameters	*n1*	
	n2	Low-order byte.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LDDGraphics(n1, n2)
```

LDEXP

LDEXP(*x, y*)

Calculates exponential (*x* * 2) to the *y*.

Parameters	*x*	The number to raise to a power of 2.
	y	The exponenetial value.
Return Value	The result of the equation (x * 2)^y.	
Comments	Requires the Math extension.	

Example

```
'Example of LDEXP(x, y)
Dim z
z = LDEXP(x, y)
```

LDoubleStrike

LDoubleStrike(*Enable*)

Sets printer’s double strike print mode.

Parameter	*Enable*	TRUE enables double strike print mode; FALSE disables double strike print mode.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LDoubleStrike(Enable)
LDoubleStrike(TRUE)
```

LDoubleWidth

LDoubleWidth(*Enable*)

Sets double width mode for `noChars` characters.

Parameter	*Enable*	TRUE enables double width print mode; FALSE disables double width print mode.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LDoubleWidth(Enable)
LDoubleWidth(TRUE)
```

LDoubleWidthN

LDoubleWidthN(*noChars*)

Sets double width mode.

Parameter *noChars* TRUE enables double width print mode; FALSE disables double width print mode.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LDoubleWidthN(Enable)
LDoubleWidthN(TRUE)
```

Left

Left(*String, n*)

Returns the leftmost *n* characters of a string.

Parameters	*String*	A string.
	N	Number of characters to return.

Return Value String containing the leftmost *n* characters of the specified string.

See Also Len, Mid operator, Right

LEmphasized

LEmphasized(*Enable*)

Sets the printer's emphasized print mode.

Parameter *Enable* TRUE enables emphasized print mode; FALSE disables emphasized print mode.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LEmphasized(Enable)
LEmphasized(TRUE)
```

Len

Len(*String*)

The Len string operator returns the length of a string.

Parameter *String* A string.

Return Value Size, in characters, of a string.

See Also Left, Mid operator, Right

LFormFeed

LFormFeed()

Sends a Form Feed character to the printer.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LFormFeed()
LFormFeed()
```

LIntnlChars

LIntnlChars(*Val*)

Sets the printer's international character set.

Parameter *Val*

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LIntnlChars(Val)
```

LLineFeed

LLineFeed(*Val*)

Moves the print head the specified number of dots.

Parameter *Val* Number of dots to feed.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LLineFeed(Val)
LLineFeed(15)
```

LLtMargin

LLtMargin(*Val*)

Sets the left margin to the specified number of characters.

Parameter *Val* Number of characters to which to set the left margin.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LLtMargin(Val)
LLtMargin(20)
```

LOG

LOG(*x*)

Calculates the natural logarithm of *x*.

Parameter *x* The number for which to calculate the natural logarithm.

Return Value The natural logarithm of *x*.

Comments Requires the Math extension.

Example

```
'Example of LOG(x)
Dim z
z = LOG(x)
```

LOG10

LOG10(*x*)

Calculates the base ten logarithm of *x*.

Parameter *x* The number for which to calculate the base ten logarithm.

Return Value The base ten logarithm of *x*.

Comments Requires the Math extension.

Example

```
'Example of LOG10(x)
Dim z
z = LOG10(x)
```

LOG2

LOG2(*x*)

Calculates the base two logarithm of *x*.

Parameter *x* The number for which to calculate the base two logarithm.

Return Value The base two logarithm of *x*.

Comments Requires the Math extension.

Example

```
'Example of LOG2(x)
Dim z
z = LOG2(x)
```

Lookup

Tables(*TableName*).Lookup(*KeyColumnName*, *DisplayColumnName*, *SearchValue*)

Finds an item in a table and returns a value from a different column of that same record.

Parameter	*TableName*	Name of the desired table.
	KeyColumnName	Name of the column to search.
	DisplayColumnName	Name of the column to return a value from.
	SearchValue	The value to search for.

Return Value The contents in the *Display* column of the matching table row if the method finds *SearchValue* in the *Key* column; or a blank string if it does not find *SearchValue*.

Comments `Lookup` is a method of the `Table` object. This function performs a linear search for the `SearchValue` in the specified `KeyColumn`, starting at the first row in the table and iterating through each row until either a matching value is found or the end or the table is reached. If a match is found, then the content of the `DisplayColumn` is returned. If not match is found, a blank string is returned. Active table filters are observed, so only visible records are searched. *New in Satellite Forms 8.*

Example

```
'Example of Lookup method
'Search table for a specific employee
'and return employee’s age
Dim age
age = Tables("Emps").Lookup("Name", "Age", "John Smith")
```

See Also Search, BinarySearch

LPageLength

LPageLength(*Length*)

Set printer's page length to the specified number of lines.

Parameter *Length* Page length in number of lines.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LPageLength(Length)
LPageLength(50)
```

LPrint

LPrint(*String*)

Prints a string.

Parameter	*String*	The string to print.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LPrint(String)
LPrint("Some text")
```

LPrintCR

LPrintCR()

Prints a carriage return. Use `LPrintLn` to print the characters specified by `LSetAutoLF`.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LPrintCR()
LPrintCR()
```

See Also LPrintLN, LSetAutoLF

LPrintDir

LPrintDir(*Val*)

Sets the `AutoLF` attribute.

Parameter *Val*

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LPrintDir(Val)
LPrintDir(0)
```

LPrintF

LPrintF(*String*, *W*, *RJustify)*

Prints the specified text in a column of the specified width.

Parameters	*String*	The string to print.
	W	Width of column in characters.
	RJustify	TRUE to right justify the column; FALSE to left justify the column.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LPrintF(String, W, RJustify)
LPrintF("Some text", 10, FALSE)
```

LPrintGraph

LPrintGraph(*mode*, *dat*, *len*)

Prints graphics in single or double density mode.

Parameters	*mode*	1 = single density print mode; 2 = double density print mode.
	dat	The graphical data to print.
	len	Number of bytes to print.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LPrintGraph(mode, dat, len)
LPrintGraph(2, theGraphic, 2048)
```

LPrintLF

LPrintLF()

Prints a Line Feed character.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LPrintLF()
LPrintLF()
```

LPrintLN

LPrintLN(*String*)

Prints text followed by a Carriage Return and a Line Feed if `AutoLF` is set to TRUE.

Parameter *String* The string to print.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LPrintLN(String)
LPrintLN("Some text")
```

See Also LSetAutoLF

LRepeatStr

LRepeatStr(*s*, *n*)

Prints the specified string the specified number of times.

Parameters *s* The string to print.

n The number of times to repeat printing.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LRepeatStr(s,n)
LRepeatStr("Some text", 10)
```

LReset

LReset()

Sends a Reset command to the printer.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LReset()
LReset()
```

LRtMargin

LRtMargin(*Val*)

Sets the right margin to the specified number of characters.

Parameter *Val* Number of characters to which to set the right margin.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LRtMargin(Val)
LRtMargin(20)
```

LSDGraphics

LSDGraphics(*n1*, *n2*)

Sets single density graphics mode for the specified number of following bytes.

Parameters *n1*

n2 Low-order byte.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LSDGraphics(n1, n2)
```

LSelectFont

LSelectFont(*Val*)

Sets the font.

Parameter *Val*

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LSelectFont(Val)
```

LSetAutoLF

LSetAutoLF(*Val*)

Sets the `AutoLF` attribute.

Parameter *Val* TRUE sets `AutoLF` to Carriage Return + Line Feed; FALSE sets `AutoLF` to Line Feed only.

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LSetAutoLF(Val)
LSetAutoLF(TRUE)
```

See Also LPrintLN

LSetGraphics

LSetGraphics(*m*, *n1*, *n2*)

Sets the printer's graphics mode and number of bytes. Supports quad density graphics.

Parameters *m*

n1

n2

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LSetGraphics(m, n1, n2)
```

LSetLine

LSetLine(*Val*)

Sets line height, usually 11 or 15 dots.

Parameter	*Val*	Line height in dots, usually 11 or 15.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LSetLine(Val)
LSetLine(15)
```

LSubscript

LSubscript(*Val*)

Sets the printer's subscript mode to the specified number of dots.

Parameter	*Val*	Number of dots to lower subscripts.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LSubscript(Val)
LSubscript(4)
```

LSuperscript

LSuperscript(*Val*)

Sets the printer's superscript mode to the specified number of dots.

Parameter	*Val*	Number of dots to raise superscripts.
Return Value	None	
Comments	Requires the Printer extension.	

Example

```
'Example of LSuperscript(Val)
LSuperscript(4)
```

LTab

LTab()

Sends a Tab character to the printer.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of LTab()
LTab()
```

LTrim

LTrim(*string*)

Trims leading spaces from input string.

Parameters	*String*	The string to trim leading spaces from.

Return Value The input string trimmed of leading spaces.

Comments Requires the Strings extension.

See Also RTrim, Trim

Example

```
'Example of LTrim
Dim strTest
strTest = LTrim("   ABCD")
'Result: strTest contains "ABCD"
```

See Also RTrim

Max

Tables(*TableName*).Max(*ColumnName*)

Returns the maximum value in a specified column in all records in a table.

Parameters *TableName* Name of a table.

ColumnName Name of a column.

Return Value Maximum value in the specified column in all records in the table.

Comments `Max` is a method of the `Table` object.

Example

```
'Example of Max method
'Output A is an edit control.
OutputA = Tables("Emps").Max("Salary")
```

See Also Sum, Min

MemoryAbout

MemoryAbout()

Displays information about the Memory extension in a dialog box.

Parameters None

Return Value None

Comments Requires the Memory extension.

Example

```
'Example of MemoryAbout()
MemoryAboutAbout()
```

MemoryAllocate

MemoryAllocate(*Size*)

Allocates dynamic memory.

Parameter *size* The number bytes to allocate.

Return Value

Comments Requires the Memory extension.

Example

```
'Example of MemoryAllocate(Size)
Memory = MemoryAllocate(128)
```

MemoryCompare

MemoryCompare(*Memory1*, *Memory2*, *Count*)

Compares two blocks of memory.

Parameters *Memory1*

Memory2

Count

Return Value

Comments Requires the Memory extension.

Example

MemoryCopy

MemoryCopy(*Memory1*, *Memory2*, *Count*)

Copies one block of dynamic memory to another.

Parameters *Memory1*

Memory2

Count

Return Value

Comments Requires the Memory extension.

Example

MemoryFree

MemoryFree(Memory)

Frees memory allocated using MemoryAllocate.

Parameter *Memory* The number bytes to allocate.

Return Value 0 if the memory is successfully freed; non-zero if the function fails.

Comments Requires the Memory extension.

Example

```
'Example of MemoryFree(Memory)
Dim bMemFreeSuccess
bMemFreeSuccess = MemoryFree(Memory1)
```

MemoryGetByte

MemoryGetByte(*Memory*, *Offset*)

Returns the value of the specified byte in memory.

Parameters *Memory* The block of dynamic memory from which to retrieve the specified byte.

Offset The offset within the block of dynamic memory from which to retrieve the specified byte.

Return Value The specified byte of memory.

Comments Requires the Memory extension.

Example

```
‘Example of MemoryGetByte(Memory, Offset)
Byte = MemoryGetByte(Memory1, 15)
```

MemoryGetString

MemoryGetString(*Memory*, *Offset*, *Count*)

Returns the value of the specified string in memory.

Parameters *Memory* The block of dynamic memory from which to retrieve the specified string.

Offset The offset within the block of dynamic memory from which to retrieve the specified string.

Count The number of characters to return.

Return Value The specified string from memory.

Comments Requires the Memory extension.

Example

```
‘Example of MemoryGetString(Memory, Offset, Count)
Byte = MemoryGetString(Memory1, 15, 10)
```

MemoryReallocate

MemoryReallocate(*Memory*, *Size*)

Resizes a block of dynamic memory.

Parameters *Memory* The block of dynamic memory to reallocate.

Size The new size for specified block of dynamic memory.

Return Value The reallocated block of memory.

Comments Requires the Memory extension.

Example

```
‘Example of MemoryReallocate(Memory, Size)
Memory1Resized = MemoryReallocate(Memory1, 25)
```

MemoryReverse

MemoryReverse(*Memory*, *Size*)

Reverses a block of dynamic memory.

Parameters *Memory* The block of dynamic memory to Reverse.

Size The size for specified block of dynamic memory.

Return Value The reversed block of memory.

Comments Requires the Memory extension.

Example

```
‘Example of MemoryReverse(Memory, Size)
Memory1Reversed = MemorySearch(Memory1, 25)
```

MemorySearch

MemorySearch(*Memory1*, *Size1*, *Memory2*, *Size2*)

Searches for a block of memory in another block of memory.

Parameters *Memory1* The block of dynamic memory in which to search.

Size1

Memory2 The block of dynamic memory to search for.

Size2

Return Value The offset of *Memory2* in *Memory1* if found; -1 if not found.

Comments Requires the Memory extension.

Example

```
‘Example of MemorySearch(Memory1, Size1, Memory2, Size2)
Offset = MemorySearch(Memory1, 15, Memory2, 10)
```

MemorySet

MemorySet(*Memory*, *Byte*, *Count*)

Sets a range of memory to the specified value.

Parameters *Memory* The block of dynamic memory to set.

Byte The value to which to set the specified range of memory.

Count The number of bytes to set to the specified value.

Return Value

Comments Requires the Memory extension.

Example

```
‘Example of MemorySet(Memory, Byte, Count)
Byte = MemorySet(Memory1, 0, 10)
```

MemorySetByte

MemorySetByte(*Memory*, *Offset*, *Byte*)

Sets the value of the specified byte in memory.

Parameters *Memory* The block of dynamic memory to set.

Offset The offset within the block of dynamic memory to which to set the specified byte value.

Byte The new byte value.

Return Value

Comments Requires the Memory extension.

Example

```
‘Example of MemorySetByte(Memory, Offset, Byte)
Byte = MemorySetByte(Memory1, 10, newByte)
```

MemorySetString

MemorySetString(*Memory*, *Offset*, *String*)

Sets the value of the specified string in memory.

Parameters *Memory* The block of dynamic memory to set.

Offset The offset within the block of dynamic memory to which to set the specified string value.

String The new string value.

Return Value

Comments Requires the Memory extension.

Example

```
‘Example of MemorySetString(Memory, Byte, String)
Byte = MemorySetString(Memory1, 15, "A new String")
```

Mid operator

Mid(*String, n, m*)

Returns *m* characters of a string beginning at character *n*.

Parameters *String* A string.

N The index of the first character to return.

m The number of characters to return.

Return Value String containing *m* characters of the specified string, beginning at character *n.*

Comments The first character in the string has an index of zero. The parameter *m* is optional. If it is omitted, the remainder of the string is returned.

See Also Left, Len, Mid statement, Right

Mid statement

Mid(*String1, n, m) = String2*

Replaces *m* characters of *String1* beginning at character *n* with the entirety of *String2*.

Parameters *String1* The string to be modified.

N The index of the first character to be replaced.

m The number of characters to replace with *String2.*

Return Value None

Comments The first character in the string has an index of zero. The parameter *m* is optional. If it is omitted, the entire remainder of the string is replaced with *String2*.

See Also Left, Len, Mid operator, Right

Min

Tables(*TableName*).Min(*ColumnName*)

Returns the minimum value in a specified column in all records in a table.

Parameters *TableName* Name of a table.

ColumnName Name of a column.

Return Value Minimum value in the specified column in all records in the table

Comments `Min` is a method of the `S` object.

See Also Max, Sum

Mod

Operand1* Mod *Operand2

Divides one value or variable by another using integer division and returns the remainder.

Parameters *Operand1* Value or variable to be divided by *Operand2*.

Operand2 Value or variable by which *Operand1* is divided.

Return Value The remainder of the integer division of *Operand1* by *Operand2.*

Comments Both operands are treated as integers if they are not integers already. Decimal places are truncated. The result is an integer.

Example

```
'Example of Mod
'InputA, InputB, and OutputA are edit controls.
OutputA = InputA Mod InputB
```

See Also]+ [add], / [divide, float], \ [divide, integer], * [multiply], - [subtract]

MODFRAC

MODFRAC(*x*)

Breaks a floating-point number into integer and fractional parts, returning the fractional part.

Parameter *x* The number to break into integer and fractional parts.

Return Value The fractional part of *x*.

Comments Requires the Math extension.

Example

```
'Example of MODFRAC(x)
Dim z
Dim x
x = 23.45634
z = MODFRAC(x)
```

MODFINT

MODFINT(*x*)

Breaks a floating-point number into integer and fractional parts, returning the integer part.

Parameter *x* The number to break into integer and fractional parts.

Return Value The integer part of *x*.

Comments Requires the Math extension.

Example

```
'Example of MODFINT(x)
Dim z
Dim x
x = 23.45634
z = MODFINT(x)
```

MoveCurrent

Tables(*TableName*).MoveCurrent

Makes the record in a form’s linked table match the record being displayed by the form.

Parameter *TableName* Name of a table.

Return Value None

Comments `MoveCurrent` is a method of the `Table` object. Use `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` to iterate through the records in a table to access or update data. These methods do not change the data displayed in the form. When you have completed iterating through all the records, use `MoveCurrent` to return the table to the record displayed on the form.

If you are updating data in the current record (the record displayed in the form) by directly updating the table, you should also update the controls on the form with the new data values. If you fail to do this, Satellite Forms saves the displayed values to the table when you leave the current record.

Note that `Move`* methods are affected by all active filters.

Example See the example for the following method, `MoveFirst`.

See Also MoveFirst, MoveLast, MoveNext, MovePrevious, RecordValid

MoveFirst

Object.MoveFirst

Moves to the first record of an object.

Parameter *Object* Name of an object.

Return Value None

Comments `MoveFirst` is a method of the `Table` object and the `Form` object and behaves differently depending on the object against which it is called.

When used with a `Form` object, `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` change both the record displayed in the form and the record in the form’s underlying table. This behavior is equivalent to the actions Goto First Record, Goto Last Record, Goto Next Record, and Goto Prev. Record.

When used with a `Table` object, `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` change the table, but do not affect the form. Use these methods to iterate through the records in a table to access or update data. When you have completed iterating through all the records, use `MoveCurrent` to return the table to the record displayed on the form.

If you are updating data in the current record (the record displayed on the form) by directly updating the table, you should also update the controls on the form with the new data values. If you fail to do this, Satellite Forms saves the displayed values to the table when you leave the current record.

Note that `Move`* methods are affected by all active filters.

Example

```
'Example of MoveFirst and MoveNext.
'Emps is a table.
'Give all employees a 10% raise.
Dim Pay
Dim NewPay
'Go to the first record.
Tables("Emps").MoveFirst
'Loop through all records.
While Tables("Emps").RecordValid = TRUE
    Pay = Tables("Emps").Fields("Salary")
    NewPay = Pay * 1.1
Tables("Emps").Fields("Salary") = NewPay
    'Go to the next record.
    Tables("Emps").MoveNext
Wend
'Update the form with the new value of the current record.
Tables("Emps").MoveCurrent Controls("Salary") = _
    Tables().Fields("Salary")
```

See Also MoveCurrent, MoveLast, MoveNext, MovePrevious, RecordValid

MoveLast

***Object*.MoveLast**

Moves to the last record of an object.

Parameter *Object* Name of an object.

Return Value None

Comments `MoveLast` is a method of the `Table` object and the `Form` object and behaves differently depending on the object.

When used with a `Form` object, `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` change both the record displayed in the form and the record in the form's underlying table. This behavior is equivalent to the actions Goto First Record, Goto Last Record, Goto Next Record, and Goto Prev. Record.

When used with a `Table` object, `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` change the table, but do not affect the form. Use these methods to iterate through the records in a table to access or update data. When you have completed iterating through all the records, use `MoveCurrent` to return the table to the record displayed on the form.

If you are updating data in the current record (the record displayed on the form) by directly updating the table, you should also update the controls on the form with the new data values. If you fail to do this, Satellite Forms saves the displayed values to the table when you leave the current record.

Note that `Move`* methods are affected by all active filters.

Example

```
'Example of MoveLast and MovePrevious
'Emps is a table.
'Salary is a column in the Emps table and an
'edit control.
'Give all employees a 10% raise.
Dim Pay
Dim NewPay
'Go to the last record.
Tables("Emps").MoveLast
'Loop for all records.
While Tables("Emps").RecordValid = TRUE
    Pay = Tables("Emps").Fields("Salary")
    NewPay = Pay * 1.1
Tables("Emps").Fields("Salary") = NewPay
'Go to the previous record.
    Tables("Emps").MovePrevious
Wend
'Update the form with the new value of the current record.
Tables("Emps").MoveCurrent
Controls("Salary") = Tables().Fields("Salary")
```

See Also MoveCurrent, MoveFirst, MoveNext, MovePrevious, RecordValid

MoveNext

Tables(*TableName*).MoveNext

Moves to the next record of an object.

Parameter *TableName* Name of a table.

Return Value None

Comments MoveNext is a method of the Table object and the Form object and behaves differently depending on the object.

When used with a Form object, MoveFirst, MoveLast, MoveNext, and MovePrevious change both the record displayed in the form and the record in the form's underlying table. This behavior is equivalent to the actions Goto First Record, Goto Last Record, Goto Next Record, and Goto Prev. Record.

When used with a Table object, MoveFirst, MoveLast, MoveNext, and MovePrevious change the table, but do not affect the form. Use these methods to iterate through the records in a table to access or update data. When you have completed iterating through all the records, use MoveCurrent to return the table to the record displayed on the form.

If you are updating data in the current record (the record displayed on the form) by directly updating the table, you should also update the controls on the form with the new data values. If you fail to do this, Satellite Forms saves the displayed values to the table when you leave the current record.

Note that Move* methods are affected by all active filters.

Example See the example of MoveFirst and MoveNext on page 410.

See Also MoveCurrent, MoveFirst, MoveLast, MovePrevious, RecordValid

MoveNextPage

Forms(*FormName*).**MoveNextPage**

Moves to the next page of a multi-page form.

Parameter *FormName* Name of a form.

Return Value None

Comments MoveNextPage is a method of the Form object. If the current page is the last page of the form, MoveNextPage moves to the next record on the first page of the form. Page numbers are zero-based.

Example

```
'Example of MoveNextPage
'Emps is a form with two pages.
If Forms("Emps").CurrentPage = 0 Then
    Forms("Emps").MoveNextPage
ElseIf Forms("Emps").CurrentPage = 1 Then
    Forms("Emps").MovePreviousPage
EndIf
```

See Also CurrentPage, MovePreviousPage

MovePrevious

Tables(*TableName*).MovePrevious

Moves to the previous record of an object.

Parameter *TableName* Name of a table.

Return Value None

Comments MovePrevious is a method of the Table object and the Form object and behaves differently depending on the object.

When used with a Form object, MoveFirst, MoveLast, MoveNext, and MovePrevious change both the record displayed in the form and the record in the form's underlying table. This behavior is equivalent to the actions Goto First Record, Goto Last Record, Goto Next Record, and Goto Prev. Record.

When used with a Table object, MoveFirst, MoveLast, MoveNext, and MovePrevious change the table, but do not affect the form. Use these methods to iterate through the records in a table to access or update data. When you have completed iterating through all the records, use MoveCurrent to return the table to the record displayed on the form.

If you are updating data in the current record (the record displayed on the form) by directly updating the table, you should also update the controls on the form with the new data values. If you fail to do this, Satellite Forms saves the displayed values to the table when you leave the current record.

Note that Move* methods are affected by all active filters.

Example See the Example of MoveLast and MovePrevious on page 411.

See Also MoveCurrent, MoveFirst, MoveLast, MoveNext, RecordValid

MovePreviousPage

Forms(*FormName*).MovePreviousPage

Moves to the previous page of a multi-page form.

Parameter *FormName* Name of a form.

Return Value None

Comments MovePreviousPage is a method of the Form object. If the current page is the first page of the form, MovePreviousPage moves to the last page of the form and displays the previous record. Page numbers are zero-based.

Example

```
'Example of MovePreviousPage
'Emps is a form with two pages.
If Forms("Emps").CurrentPage = 0 Then
    Forms("Emps").MoveNextPage
ElseIf Forms("Emps").CurrentPage = 1 Then
    Forms("Emps").MovePreviousPage
EndIf
```

See Also CurrentPage, MoveNextPage

MoveRecord

Tables(*TableName*).MoveRecord(*ToRec, FromRec*)

Removes a record at *FromRec* and reinserts it before *ToRec*.

Parameters	*TableName*	Name of a table
	ToRec	Index of the record before which *FromRec* is to be placed.
	FromRec	Index of the record to move.

Return Value None

Comments `MoveRecord` is a method of the `Table` object. The indexes for *FromRec* and *ToRec* are both zero-based.

MsgBox

MsgBox(*MessageText*)

Displays a dialog box with the specified message and an OK button.

Parameter *MessageText* Text of message to be displayed in dialog box.

Return Value None

Comments `MsgBox` is a method of the `App` object.

Example

```
'Example of the MsgBox method
'InputA is an edit control.
MsgBox("The answer is " & InputA)
```

See Also Prompt,

Not [bitwise]

Not *numerical expression*

This operator, when used with a numerical expression, performs a bitwise `Not` operation on its argument.

Parameter *Numerical expression* Numerical value or expression to which the Not operation ise applied.

Return Value The bitwise `Not` of the specified number.

Example

```
'Example of Not
'InputA and OutputA are edit controls.
OutputA = Not Int(InputA)
```

See Also And [bitwise]And [logical], Or [bitwise], Xor [bitwise]

Not [logical]

Not *logical expression*

Negates a logical value or variable.

Parameter *Expression* Logical value or variable to be negated.

Return Value TRUE if *expression* evaluates to FALSE; FALSE if *expression* evaluates to TRUE.

Comments Note that Xor, And, Or, and Not only perform Boolean operations if both conditions are Boolean. Otherwise, they perform bitwise operations on their operands.

Example

```
'Example of Not
'InputA, InputB, and OutputA are edit controls.
If InputA = InputB Then
    OutputA = "A=B"
ElseIf Not (InputA = InputB) Then
    OutputA = "A<>B"
EndIf
```

See Also And [bitwise]And [logical], Or [logical], Xor [logical]

OnClick

Occurs for a control when the user taps the control.

Comments OnClick is an event of the Control and Extension objects.

Use OnClick to perform logic that executes whenever the user taps a control. Controls that support the OnClick event include buttons, check boxes, drop lists, list boxes, radio buttons, and SFX Custom controls (if supported by the control).

OnClick fires after the control executes its intrinsic action – check boxes check/uncheck themselves before firing an OnClick event.

OnClick is useful for many functions. For example, you can place a calculate button on an order form that runs a script that sums the order based on the customer’s discount schedule and then adds applicable taxes.

OnKey

Occurs when the user presses one of the plastic keys or silk-screened buttons, or enters a Graffiti stroke on a handheld device.

Comments Use this event to perform special actions when the user presses a particular key or enters a particular stroke in the Graffiti area. Call GetLastKey inside your OnKey event handler to determine which key the user pressed.

If you exit the handler with the Exit keyword, the key is passed on to the operating system as if you did not have an OnKey event handler. If, however, you exit the handler with the Fail keyword, the key is discarded as if it were never entered.

See Also GetLastKey

OnPenDown

Occurs when the stylus touches the screen.

Comments Typically, code that tracks the pen is placed in this handler. Call GetPenStatus in a loop and perform your processing. When GetPenStatus returns FALSE, exit the loop and the event handler.

See Also GetPenStatus, OnPenUp

OnPenUp

Occurs when the user lifts the stylus from the screen.

Comments This event is not usually used. Applications that track the stylus usually do so with the `OnPenDown` event described above.

See Also GetPenStatus, OnPenDown

OnTimer

When the timer is on, an `OnTimer` event occurs every period of the timer.

Comments The timer event is usually used for performing repetitive tasks without tying up the CPU. If use a loop to perform a repetitive task, call `Delay`, and then loop again, the CPU is tied up for the duration of the loop and the handheld cannot respond to any user interaction.

If instead, you call `SetTimer` to turn on the timer with the desired frequency and then perform one iteration of the repetitive task in the `OnTimer` event handler each time it is called, the CPU is free to interact with the user while the handler is not running. When done with the timer, call `KillTimer` to turn it off.

See Also KillTimer, SetTimer

OnValidate

Occurs when a form's data is validated.

Comments Use `OnValidate` to perform logic that should be executed when a form's data is validated.

`OnValidate` fires when a user navigates to another record or page in the current form, jumps to another form, applies a filter to the form's linked table, or exits the current application.

When a form closes, first `OnValidate` fires and then `BeforeClose` fires. If the script associated with `OnValidate` fails, `BeforeClose` never fires and the form is not exited.

`OnValidate` is useful for performing complex data validation. For example, you might total a customer's order and ensure that the total is within the customer's credit limit. If the order total exceeds the credit limit, you can fail the `OnValidate` event and prompt the user to adjust the order information.

You can prevent the user from exiting an application with incomplete data by checking the data in the `OnValidate` handler and using the `Fail` keyword if you want to prevent the application from exiting.

See Also BeforeClose

OpenPort

OpenPort()

Opens the serial port using the settings specified with SetPort. Opening the serial port uses more battery power.

Parameters None

Return Value None

Comments Requires the Printer extension.

Example

```
'Example of OpenPort()
OpenPort()
```

See Also ClosePort

Or [bitwise]

Number1* Or *Number2

Performs a bitwise `Or` operation between the operands.

Parameters *Number1* First operand.

Number2 Second operand.

Return Value The result of a bitwise `Or` of the two operands.

Example

```
'Example of bitwise Or
'InputA is an edit control.
'Set bit 4 (mask = 10 hex) in number.
OutputA = InputA Or &H10
```

See Also And [bitwise]And [logical], Not [bitwise], Xor [bitwise]

Or [logical]

Condition1* Or *Condition2

Joins two conditions where either condition must evaluate to TRUE for the statement to evaluate to TRUE.

Parameters *Condition1* First condition to be evaluated.

Condition2 Second condition to be evaluated.

Return Value TRUE if *Condition1* evaluates to TRUE or *Condition2* evaluates to TRUE; FALSE if both *Condition1* and *Condition2* evaluate to FALSE.

Comments If both *Condition1* and *Condition2* evaluate to TRUE, the statement evaluates to TRUE. Each condition is evaluated individually, then the `Or` statement is performed. Note that `Xor`, `And`, `Or`, and `Not` only perform Boolean operations if both conditions are Boolean. Otherwise, they perform bitwise operations on their operands.

Example

```
'Example of Or
'InputA, InputB, and OutputA are edit controls.
If InputA > 10 Or InputB > 10 Then
    OutputA = "TRUE"
Else
    OutputA = "FALSE"
EndIf
```

See Also And [bitwise]And [logical], Not [logical], Xor [logical]

Pad

Pad(*length, string, padchar*)

Pads a string with defined character to defined length. The pad character is added to the beginning of the string to make a final string of the defined length.

Parameters *Length* Desired length to pad string out to.

String Source string to be padded

PadChar Character to left pad the string with.

Return Value String padded with padchar out to specified length.

Comments Requires the Strings extension.

Example

```
'Example of Pad
Dim strTest
strTest = Pad(8, “12345”, “0”)
'Result: strTest contains “00012345”
```

PBD_About

PBD_About()

Displays information about the extension in a dialog box.

Parameters None

Return Value None

Comments Requires the Puma Beam DB extension.

Example

```
'Example of PBD_About()
PBD_About()
```

PBD_BeamDb

PBD_BeamDb(*Local_ID*, *Card_No*, *File_Name*, *Desc_Name*)

Sends a Palm database file to another device using the IR port.

Parameters

Local_ID	
Card_No	
File_Name	The name of the database file to be transferred.
Desc_Name	Descriptive name for the user.

Return Value None

Comments Requires the Puma Beam DB extension.

See Also PBD_SendDbByName

PBD_SendDbByName

PBD_SendDbByName(*File_Name*, *Desc_Name*)

Sends Palm Db specified by name, giving user option to select transport method (Beam, Bluetooth, SMS, VersaMail, etc.).

Parameters

File_Name	The name of the database file to be transferred.
Desc_Name	Descriptive name for the user.

Return Value None

Comments Requires the Puma Beam DB extension.

See Also PBD_BeamDb

PBD_Version

PBD_Version()

Returns the version number of this extension.

Parameters None

Return Value The version number of this extension.

Comments Requires the Puma Beam DB extension.

PDM_About

PDM_About()

Displays information about the extension in a dialog box.

Parameters None

Return Value None

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_About()
PDM_About()
```

PDM_DeleteDb

PDM_DeleteDb(*Local_ID*, *Card_No*)

Deletes the specified database.

Parameters *Local_ID*
Card_No

Return Value None

Comments Requires the Puma Data Manager extension.

Example `'Example of PDM_DeleteDb(Local_ID, Card_No)`

PDM_GetDbAppInfoID

PDM_GetDbAppInfoID(*Local_ID*)

Returns the app info ID of the `LastDb` cache (Local ID).

Parameter *Local_ID*

Return Value The app info ID of the `LastDb` cache (Local ID).

Comments Requires the Puma Data Manager extension.

Example `'Example of PDM_GetDbAppInfoID(Local_ID)`

PDM_GetDbAttributes

PDM_GetDbAttributes()

Returns attributes of the `LastDb` cache.

Parameters None

Return Value The attributes of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example `'Example of PDM_GetDbAttributes()`

PDM_GetDbBckUpDate

PDM_GetDbBckUpDate()

Returns the last back up date of the `LastDb` cache.

Parameters None

Return Value The last back up date of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example `'Example of PDM_GetDbBckUpDate()`

PDM_GetDbBckUpDateStr

PDM_GetDbBckUpDateStr()

Returns the last back up date of the `LastDb` cache as a string.

Parameters None

Return Value The last back up date of the `LastDb` cache as a string.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbBckUpDateStr()
Dim BackupDateStr
BackupDateStr = PDM_GetDbBckUpDateStr()
```

PDM_GetDbCardNo

PDM_GetDbCardNo()

Returns the card number of the `LastDb` cache.

Parameters None

Return Value The card number of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbCardNo()
Dim DbCardNo
DbCardNo = PDM_GetDbBckUpDateStr()
```

PDM_GetDbCrDate

PDM_GetDbCrDate()

Returns the creation date of the `LastDb` cache.

Parameters None

Return Value The creation date of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbCrDate()
```

PDM_GetDbCrDateStr

PDM_GetDbCrDate()

Returns the creation date of the `LastDb` cache as a string.

Parameters None

Return Value The creation date of the `LastDb` cache as a string.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbCrDateStr()
Dim DbCreateDateStr
DbCreateDateStr = PDM_GetDbCrDateStr()
```

PDM_GetDbCreatorID

PDM_GetDbCreatorID()

Returns the creator ID of the `LastDb` cache.

Parameters None

Return Value The creator ID of the `LastDb` cache as a string.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbCreatorID()
Dim DbCreatorID
DbCreatorID = PDM_GetDbCreatorID()
```

PDM_GetDbDataBytes

PDM_GetDbDataBytes()

Returns the number of bytes in the data portion of the `LastDb` cache.

Parameters None

Return Value The number of bytes in the data portion of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbDataBytes()
Dim DbDataBytes
DbDataBytes = PDM_GetDbDataBytes()
```

PDM_GetDbModDate

PDM_GetDbModDate()

Returns the modification date of the `LastDb` cache.

Parameters None

Return Value The modification date of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbModDate()
```

PDM_GetDbModDateStr

PDM_GetDbModDateStr()

Returns the modification date of the `LastDb` cache as a string.

Parameters None

Return Value The modification date of the `LastDb` cache as a string.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbModDateStr()
Dim DbModDateStr
DbModDateStr = PDM_GetDbModDateStr()
```

PDM_GetDbModNum

PDM_GetDbModNum()

Returns the number of modifications to the `LastDb` cache.

Parameters None

Return Value The number of modifications to the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbModNum()
Dim DbModNum
DbModNum = PDM_GetDbModNum()
```

PDM_GetDbName

PDM_GetDbName()

Returns the name of the last database loaded.

Parameters None

Return Value The name of the last database loaded.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbName()
Dim DbName
DbName = PDM_GetDbName()
```

PDM_GetDbNumRecords

PDM_GetDbNumRecords()

Returns the number of records contained in the `LastDb` cache.

Parameters None

Return Value The number of records contained in the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbNumRecords()
Dim DbNumRecs
DbNumRecs = PDM_GetDbNumRecords()
```

PDM_GetDbSortInfoID

PDM_GetDbSortInfoID()

Returns the sort info ID of the LastDb cache.

Parameters None

Return Value The sort info ID of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbSortInfoID()
Dim DbSortInfoID
DbSortInfoID = PDM_GetDbSortInfoID()
```

PDM_GetDbTotalBytes

PDM_GetDbTotalBytes()

Returns the number of bytes in the `LastDb` cache.

Parameters None

Return Value The number of bytes in the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbTotalBytes()
Dim DbTotalBytes
DbTotalBytes = PDM_GetDbTotalBytes()
```

PDM_GetDbType

PDM_GetDbType()

Returns the type of the `LastDb` cache.

Parameters None

Return Value The type of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbType()
Dim DbType
DbType = PDM_GetDbType()
```

PDM_GetDbVersion

PDM_GetDbVersion()

Returns the version number of the `LastDb` cache.

Parameters None

Return Value The version number of the `LastDb` cache.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetDbVersion()
Dim DbVersion
DbVersion = PDM_GetDbVersion()
```

PDM_GetLastError

PDM_GetLastError()

Returns the last error code.

Parameters None

Return Value The last error code.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetLastError()
Dim LastError
LastError = PDM_GetLastError()
```

PDM_GetNextDb

PDM_GetNextDb()

Returns the local ID of the next database that matches the search criteria. Consecutive calls to this method traverses all databases matching the search criteria.

Parameters None

Return Value The local ID of the next database that matches the search criteria.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetNextDb()
Dim NextDbID
NextDbID = PDM_GetNextDb()
```

PDM_GetNumberOfMatchingDb

PDM_GetNumberOfMatchingDb(*CreatorID*, *TypeID*, *Version*)

Returns the number of databases that match the search criteria.

Parameters *CreatorID*

TypeID

Version

Return Value The number of databases that match the search criteria.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_GetNumberOfMatchingDb(CreatorID, TypeID, Version)
```

PDM_LoadDb

PDM_LoadDb(*Local_ID*, *Card_No*)

Loads the specified database into the `LastDb` cache.

Parameters *Local_ID*
Card_No

Return Value None

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_LoadDb(Local_ID, Card_No)
```

PDM_NewDbIterator

PDM_NewDbIterator(*CreatorID*, *TypeID*, *Version*)

Starts a new search.

Parameters *CreatorID*
TypeID
Version

Return Value None

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_NewDbIterator(CreatorID, TypeID, Version)
```

PDM_SetDbAttributes

PDM_SetDbAttributes(*LocalID*, *Card_No*, *New_Attributes*)

Sets the attributes of the specified database.

Parameters *LocalID*
Card_No
New_Attributes

Return Value None

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_SetDbAttributes(LocalID, Card_No, New_Attributes)
```

PDM_Version

PDM_Version()

Returns the version number of this extension.

Parameters None

Return Value The version number of this extension.

Comments Requires the Puma Data Manager extension.

Example

```
'Example of PDM_Version()
Dim DbVersion
DbVersion = PDM_Version()
```

PEM_About

PEM_About()

Displays information about the extension in a dialog box.

Parameters None

Return Value None

Comments Requires the Puma Error Manager extension.

Example

```
'Example of PEM_About()
PEM_About()
```

PEM_GetErrorString

PEM_GetErrorString(Error_Number)

Converts an error code to a string.

Parameter *Error_Number* The error code for which to retrieve a descriptive string.

Return Value The string description for the specified error code..

Comments Requires the Puma Error Manager extension.

Example

```
'Example of PEM_GetErrorString(Error_Number)
Dim x
x = PEM_GetErrorString(134)
```

PEM_Version

PEM_Version()

Returns the version number of this extension.

Parameters None

Return Value The version number of this extension.

Comments Requires the Puma Error Manager extension.

Example

```
'Example of PEM_Version()
Dim x
x = PEM_Version()
```

Popup

ControlName.Popup

For droplist controls, this method pops up the selection list as if the user had tapped on the droplist control. For edit and paragraph controls that have the AutoKeyboard setting enabled, this method pops up the defined automatica keyboard for that control.

Parameter *ControlName* Name of a control.

Return Value None

Comments `Popup` is a method of the `Control` object. It enables you to pop up a droplist selection list or an edit/paragraph control automatica keyboard under script control.

Position

Tables(*TableName*).Position

Returns the row number of the current row in the table or can be assigned to set the table to a specific row.

Parameter *TableName* Name of a table.

Return Value Zero-based number of the current row in the table if used to access the position, or none if used to set the position.

Comments `Position` is a property of the `Table` object. This property is not normally used. Navigation through records of a table should use the `Move`* table methods whenever possible.

Note: The `Position` property does **not** take into account any active filters. By successively incrementing a value starting at zero and using it to set the `Position` property of a table, you always visit all records of the table.

POW

POW(*x, y*)

Calculates exponential *x* to the *y*.

Parameters *x* The number to raise to the power *y*.

y The exponential value.

Return Value The result of x^y.

Comments Requires the Math extension.

Example

```
'Example of POW(x, y)
Dim z
z = POW(x, y)
```

PreviousForm

Forms().PreviousForm

Returns to the previous form.

Parameter *FormName* Name of the current form (optional, not needed).

Return Value None

Comments `PreviousForm` is a method of the `Form` object.

PreviousForm returns to the previous form, just like using a control with the ReturnToPreviousForm action.

See Also Show

Process

Extension.Process()

Call from a Timer event to process automatic events.

Parameters None

Return Value TRUE if a scan was processed for Symbol Integrated Scanner.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of Process()
BarCode1.Process()
```

Prompt

Prompt(*MessageText*)

Displays the specified message in a dialog box with OK and CANCEL buttons.

Parameter *MessageText* Text of message to be displayed in dialog box.

Return Value 1 if the user taps the OK button;
0 if the user taps the CANCEL button

Comments `Prompt` is a method of the `App` object.

Example

```
'Example of the Prompt method
Dim x
x = Prompt("Do you want to continue?")
If x = 1 Then
    MsgBox("OK, we will continue.")
Else
    MsgBox("I’m sorry you want to quit.")
EndIf
```

See Also MsgBox, PromptCustom

PromptCustom

PromptCustom(***strTitle, strMessage, strButton1, strButton2, strButton3***))

Displays a prompt dialog containing custom buttons and a custom message. This prompt dialog is dismissed when any of the buttons are clicked.

Parameters

strTitle Title of the prompt dialog

strMessage Text of message to be displayed in dialog box.

strButton1 Text of button no 1 (if no text is specified, button 1 will be invisible)

strButton2 Text of button no 2 (if no text is specified, button 2 will be invisible)

strButton3 Text of button no 3 (if no text is specified, button 3 will be invisible)

Return Value 0 if the user taps button1;
1 if the user taps button2;

2 if the user taps button3

Comments `PromptCustom` is a method of the `App` object.

Example

```
'Example of the PromptCustom method
Dim x
x = PromptCustom(“Select Option”,"Choose the desired option?",
“Opt 1”, “Opt 2”, “Opt 3”)
If x = 0 Then
    MsgBox("Option 1 selected")
ElseIf x = 1 Then
    MsgBox("Option 2 selected")
Else
    MsgBox("You selected Option 3")
EndIf
```

See Also Prompt, MsgBox

QuickSort

Tables(***TableName***).QuickSort(***ColumnName, Direction)***

Sorts the records in the specified table using the specified column as the key in ascending or descending order.

Parameters

TableName Name of a table.

ColumnName Name of a column.

Direction Sort order. Use TRUE for ascending and FALSE for descending.

Return Value None

Comments `QuickSort` is a method of the `Table` object. This method is faster than `InsertionSort`, but it does not preserve the relative order of the previous sort (presumably sorted on a different key). For example, if you sorted on NAME and then sorted by AGE, the table would be sorted by AGE and the previous sort on NAME would not be preserved, that is, records within the same age would not be sorted by NAME.

Example

```
'Example of QuickSort method
'Sort table by employee name.
Tables("Emps").QuickSort("Name", TRUE)
```

See Also InsertionSort

Quit

Quit

Exits from the application and returns to the app launcher.

Parameters None

Return Value None

Comments Use `Quit` to close your application and return to the app launcher. Form validation events and the BeforeAppEnd event wil be called just as with a user-initiated exit.

Example

```
'Close the application now
Quit
```

ReadOnly

ControlName.ReadOnly

Returns or sets the value of a control’s read-only attribute.

Parameter *ControlName* Name of a control.

Return Value TRUE if data in the object is read-only; FALSE if data in the object can be edited.

Comments `ReadOnly` is a property of the `Control` object. You can also change a control’s read-only property by setting this property. In that case, the function does not return a value.

Example

```
'Example of ReadOnly property
'InputA is an edit control.
'Make data in InputA non-editable.
If InputA.ReadOnly = FALSE Then
    InputA.ReadOnly = TRUE
EndIf
```

RecordAdv

Extension.RecordAdv(*str*)

Sets the Record Advance mode.

Parameter *str*

"Off" Never changes the record. You must do so manually using the `AFTERSCAN` property.

"On" Advances to last record and stops.

"AlwaysCrt" Creates a new record after last Edit control.

"CrtAtEnd" Advance through the records, create new after last record.

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of RecordAdv(str)
BarCode1.RecordAdv("On")
```

RecordValid

Tables(*TableName*).RecordValid

Indicates whether any of the `Move*` methods have moved to a valid record in the table.

Parameters *TableName* Name of a table.

Return Value TRUE if the current record is valid; FALSE if it is not.

Comments `RecordValid` is a read-only property of the `Table` object. Use `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` to iterate through the records in a table to access or update data. These methods do not change the data displayed on the form. When are finished iterating through the records, use `MoveCurrent` to return the table to the record displayed on the form.

Note that `Move*` methods are affected by all active filters.

This property is FALSE if there are no records in the table or there are no more records that satisfy all active filters.

Example

```
'Example of RecordValid
'Emps is a table.
'Salary is a column in the Emps table and an edit control.
'Give all employees a 10% raise.
Dim Pay
Dim NewPay
'Go to the last record.
Tables("Emps").MoveLast
'Loop for all records.
While Tables("Emps").RecordValid = TRUE
    Pay = Tables("Emps").Fields("Salary")
    NewPay = Pay * 1.1
    Tables("Emps").Fields("Salary") = NewPay
    'Go to the previous record.
    Tables("Emps").MovePrevious
Wend
'Update the form with the new value of the
'current record.
Tables("Emps").MoveCurrent
Controls("Salary") = Tables().Fields("Salary")
```

See Also MoveCurrent, MoveFirst, MoveLast, MoveNext, MovePrevious

Refresh

Forms(*FormName*).Refresh

Saves the contents of the form’s controls to the underlying table and then calls `Requery` to reload that data and to redraw the screen.

Parameter *FormName* Name of a form.

Return Value None

Comments `Refresh` is a method of the `Form` object.

RemoveAllFilters

RemoveAllFilters

Clears (removes) all active filters so that all records are visible.

Parameter *None*

Return Value None

Comments `RemoveAllFilters` is a method of the `App` object.

Example

```
'Example of RemoveAllFilter
RemoveAllFilters
```

See Also AddFilter, RemoveFilter

RemoveFilter

Tables(*TableName*).RemoveFilter(*ColumnName*)

Clears a filter in a table.

Parameter	*TableName*	Name of a table.
	ColumnName	Name of the column to remove the filter condition from.

Return Value None

Comments `RemoveFilter` is a method of the `Table` object.

Example

```
'Example of RemoveFilter
'Emps is a table.
'Salary is a columns in the Emps table.
Tables("Emps").RemoveFilter("Salary")
```

See Also AddFilter, RemoveAllFilters

RemoveRecord

Tables(*TableName*).RemoveRecord(*RecordNumber*)

Removes a record from a table. It differs from `DeleteRecord` in that it deletes the record immediately, not at the next HotSync. Only use this method if the table you are operating on will not by synchronizeed to a table on the desktop/server, otherwise the deletion will not be detected. This method does not prompt the user to confirm the deletion.

Parameters	*TableName*	Name of a table.
	RecordNumber	Zero-based number of the row to be deleted.

Return Value None

Comments `RemoveRecord` is a method of the `Table` object.

Example

```
'Example of Remove Record 'method
Dim count, i
'Get number of records in the table
count = Tables("emp").Count()
'Iterate through all records and remove
'the first record in the table.
for i=0 to count-1
    Tables("emp").RemoveRecord(0)
next i
```

See Also CreateRecord, DeleteRecord

Repaint

Forms(*FormName*).Repaint

Redraws the form and the controls on the screen.

Parameter *FormName* Name of a form.

Return Value None

Comments `Repaint` is a method of the `Form` object.

Example

```
'Redraw this form.
Forms().Repaint
```

Replace

Replace(*string*, *originalsubstring, newsubstring*)

Replace a substring with another substring, using a case sensitive comparison.

Parameters	*string*	The string to trim trailing spaces from.
	old	The substring to be replaced.
	new	The new substring to replace the old substring with.

Return Value The modified string with the substring replaced.

Comments Requires the Strings extension.

Example

```
'Example of Replace
strTest = Replace("The Chicken was chicken", "Chicken", "Fox")
'returns "The Fox was chicken"
```

See Also LTrim, Trim, RTrim

Requery

Forms(*FormName*).Requery

Reloads controls on the form from the underlying table and then calls `Repaint`.

Parameter *FormName* Name of a form.

Return Value None

Comments `Requery` is a method of the `Form` object. Any data currently in screen controls that has not been saved to the underlying tables is lost. This operation, in essence, is a discard current data and reload operation.

ResetCtrls

Extension.ResetCtrls()

Sets the index to the first control and cancels Record Advance mode.

Parameters None

Return Value Always TRUE for the Symbol Integrated Scanner control. None for Bar Code Reader control.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of ResetCtrls()
BarCode1.ResetCtrls()
```

See Also RecordAdv

RestorePrevColor

RestorePrevColor()

Restores the previous color scheme.

Parameters None

Return Value None

Comments Applies only to the Color Graphics and Graphics extensions.

Example
```
'Example of RestorePrevColor()
RestorePrevColor()
```

Right

Right(*String, n*)

Returns the rightmost *n* characters of a string.

Parameters *String* A string.

n The number of characters to return.

Return Value String containing the rightmost *n* characters of the specified string.

See Also Len, Left, Mid operator

RestoreScreenDepth

RestoreScreenDepth()

Restores Palm screen depth to the value before using `SetMaxScreenDepth`.

Parameters None

Return Value None

Comments Requires the PxScreen Tool extension.

Caution: You must call this method in the `BeforeAppEnd` global script. Otherwise, using a menu crashes the handheld device.

Example
```
'Example of RestoreScreenDepth()
RestoreScreenDepth()
```

See Also SetMaxScreenDepth

RM_AttachCurrentResource

RM_AttachCurrentResource()

Attaches the current resource to the database.

Parameters None

Return Value 0 if successful; error code if the method fails.

Comments Requires the Puma Resource Manager extension.

Example
```
'Example of RM_AttachCurrentResource()
Dim AttachSuccess
AttachSuccess = RM_AttachCurrentResource()
```

RM_CloseDatabase

RM_CloseDatabase()

Closes the current database.

Parameters None

Return Value 0 if successful; error code if the method fails.

Comments Requires the Puma Resource Manager extension.

Example
```
'Example of RM_CloseDatabase()
Dim CloseSuccess
CloseSuccess = RM_CloseDatabase()
```

RM_DetachedResToCurRes

RM_DetachedResToCurRes()

Moves the detached resource to the current resource.

Parameters None

Return Value None

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_DetachedResToCurRes()
RM_DetachedResToCurRes()
```

RM_DetachResource

RM_DetachResource()

Detaches the resource and stores the current pointer.

Parameters None

Return Value 0 if successful; error code if the method fails.

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_DetachResource()
Dim DetachSuccess
DetachSuccess = RM_DetachResource()
```

RM_FindResource

RM_FindResource(*ResType*, *ResourceID*)

Finds the current resource by resource type and ID.

Parameters *ResType*

ResourceID

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_FindResource(ResType, ResourceID)
```

RM_FindResourceByIndex

RM_FindResourceByIndex(*ResType*, *ResourceIndex*)

Searches for the resource by type and index number.

Parameters *ResType*

ResourceIndex

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_FindResourceByIndex(ResType, ResourceIndex)
```

RM_Get1Resource

RM_Get1Resource(*ResType*, *ResourceIndex*)

Returns the resource from the most currently opened database.

Parameters *ResType*

ResourceIndex

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_Get1Resource(ResType, ResourceIndex)
```

RM_GetLastErrorNumber

RM_GetLastErrorNumber()

Returns the last Resource Manager error code.

Parameters None

Return Value The last Resource Manager error code.

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_GetLastErrorNumber()
Dim LastError
LastError = RM_GetLastErrorNumber()
```

RM_GetResource

RM_GetResource(*ResType*, *ResourceID*)

Returns the resource specified by the type and ID.

Parameters *ResType*
ResourceID

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_GetResource(ResType, ResourceID)
```

RM_GetResourceByIndex

RM_GetResourceByIndex(*ResType*, *ResourceIndex*)

Returns the resource specified by index..

Parameter *ResourceIndex*

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_GetResourceByIndex(ResourceIndex)
```

RM_LockRes

RM_LockRes()

Locks the current resource in memory.

Parameters None

Return Value None

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_LockRes()
RM_LockRes()
```

RM_NewResource

RM_NewResource(*ResType*, *ResourceID*, *Size*)

Adds a new resource to the open database.

Parameters *ResType*
ResourceID
Size

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_NewResource(ResType, ResourceID, Size)
```

RM_NumResource

RM_NumResource()

Returns the number of resources in the current database.

Parameters None

Return Value The number of resources in the current database.

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_NumResource()
Dim NumResources
NumResources = RM_NumResource()
```

RM_OpenDatabase

RM_OpenDatabase(*Name*, *Card_No*, *mode*)

Open the specified resource database.

Parameters *Name*
Card_No
mode

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_OpenDatabase(Name, Card_No, mode)
```

RM_OpenDBNoOverlay

RM_OpenDBNoOverlay(*Name*, *Card_No*, *mode*)

Open the specified resource database with no modifications to the overlay.

Parameters *Name*
Card_No
mode

Return Value

Comments Requires the Puma Resource Manager extension and Palm OS 3.5 or greater.

Example

```
'Example of RM_OpenDBNoOverlay(Name, Card_No, mode)
```

RM_PassResPtr

RM_PassResPtr()

Returns a pointer to the memory block of a locked resource.

Parameters None

Return Value A pointer to the memory block of a locked resource.

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_PassResPtr()
Dim ResPtr
ResPtr = RM_PassResPtr()
```

RM_ReleaseResource

RM_ReleaseResource()

Releases the current resource.

Parameters None

Return Value 0 if successful; error code if the method fails.

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_ReleaseResource()
Dim ReleaseSuccess
ReleaseSuccess = RM_ReleaseResource()
```

RM_RemoveResource

RM_RemoveResource(*ResourceIndex*)

Removes the specified resource from the current database.

Parameter *ResourceIndex*

Return Value 0 if successful; error code if the method fails.

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_RemoveResource(ResourceIndex)
```

RM_ResizeResource

RM_ResizeResource(*Size*)

Resizes the current resource.

Parameter *Size*

Return Value

Comments Requires the Puma Resource Manager extension.

Example

```
'Example of RM_ResizeResource(Size)
```

RM_ResourceInfo

RM_ResourceInfo(*ResourceIndex*, *Info_to_Return*)

Returns information about the specified resource.

Parameters *ResourceIndex*
Info_to_Return

Return Value Information about the specified resource.

Comments Requires the Puma Resource Manager extension and Palm OS 3.5 or greater.

Example

```
'Example of RM_ResourceInfo(ResourceIndex, Info_to_Return)
```

RM_ResTypeInfo

RM_ResTypeInfo(*ResourceIndex*)

Returns the resource type.

Parameter *ResourceIndex*

Return Value The resource type.

Comments Requires the Puma Resource Manager extension.

Example `'Example of RM_ResTypeInfo(ResourceIndex)`

RM_SearchResource

RM_SearchResource(*ResType*, *ResourceID*)

Searches for a resource by type and local ID.

Parameters *ResType*
ResourceID

Return Value

Comments Requires the Puma Resource Manager extension and Palm OS 3.5 or greater.

Example `'Example of RM_SearchResource(ResType, ResourceID)`

RM_SetResourceInfo

RM_SetResourceInfo()

Sets the resource info for the current resource.

Parameters None

Return Value 0 if successful; error code if the method fails.

Comments Requires the Puma Resource Manager extension.

Example `'Example of RM_SetResourceInfo()`

RM_UnlockRes

RM_UnlockRes()

Unlocks the current resource. Use only after using `RM_LockRes`.

Parameters None

Return Value

Comments Requires the Puma Resource Manager extension.

Example `'Example of RM_UnlockRes()`

See Also RM_LockRes

ROUND

ROUND(*x, decimal*)

Rounds a number to nearest decimal place specified.

Parameters *x* The number to round off.

decimal The number of decimal places to which to round.

Return Value The rounded number.

Comments Requires the Math extension.

Example

```
'Example of ROUND(x, decimal)
Dim z
Dim y
Dim x
x = 2.3475947495
y = 3
z = ROUND(x, y)
```

RTrim

RTrim(*string*)

Trims trailing spaces from input string.

Parameters *String* The string to trim trailing spaces from.

Return Value The input string trimmed of trailing spaces.

Comments Requires the Strings extension.

Example

```
'Example of RTrim
Dim strTest
strTest = RTrim("ABCD   ")
'Result: strTest contains "ABCD"
```

See Also LTrim, Trim

ScanAvail

Extension.ScanAvail()

Returns the integer byte count in the receive buffer.

Parameters None

Return Value Number of bytes in the receive buffer.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Caution: You must call `EnableScanner` before using this method with the Symbol Integrated Scanner. Otherwise, a fatal exception occurs.

Example

```
'Example of ScanAvail()
Dim x
x = BarCode1.ScanAvail()
```

See Also EnableScanner

Scroll

Controls(*ControlName*).Scroll(*NumLines*)

The `Scroll` method is only available for Paragraph controls. It scrolls the control up or down by *NumLines* lines.

Parameters *ControlName* Name of a control.

NumLines Number of lines.

Return Value None

Comments `Scroll` is a method of the `Control` object. If *NumLines* is positive, the control scrolls down – the text goes up. If *NumLines* is negative, the control scrolls up.

Search

Tables(*TableName*).Search(*ColumnName*, *SearchValue*)

Finds an item in a table.

Parameter *TableName* Name of the desired table.

ColumnName Name of the column to search.

SearchValue The value to search for.

Return Value The row number in the table if the method finds *SearchValue*; or 65535 (0xFFFF in hexadecimal) if it does not find *SearchValue*.

Comments `Search` is a method of the `Table` object. This function performs a linear search for the `SearchValue` in the specified `Column`, starting at the first row in the table and iterating through each row until either a matching value is found or the end or the table is reached. Active table filters are observed, so only visible records are searched. The table does not need to be sorted for the `Search` function to work. However, if your table is sorted on the search column, consider using the BinarySearch function instead, which is much faster when dealing with large numbers of records. *New in Satellite Forms 8.*

Example

```
'Example of Search method
'Search table for a specific employee.
Dim RowNum
RowNum = Tables("Emps").Search("Name", "John Smith")
```

See Also BinarySearch, Lookup

Seed_Val

Seed_Val()

Returns the seed value entered for use with `SRand48`.

Parameters None

Return Value The seed value entered for use with `SRand48`.

Comments Requires the Random Number Generator extension.

Example

```
'Example of Seed_Val()
Dim z
z = Seed_Val()
```

See Also Seed48, SRand48

Seed_Val16V

Seed_Val16V(*value*)

Returns 16 bits of the three orders of the 48-bit seed value used with `Seed48`.

Parameter *value* Must be 1, 2 or 3, designating the desired 16-bit order.

Return Value The specified 16-bit order.

Comments Requires the Random Number Generator extension.

Example

```
'Example of Seed_Val16V(value)
Dim z
z = Seed_Val16V(3)
```

See Also Seed48

Seed48

Seed48(*X1*, *X2*, *X3*)

Sets the 48-bit seed value.

Parameters	*X1*	The first 16 bits of the seed value.
	X2	The second 16 bits of the seed value.
	X3	The last 16 bits of the seed value.

Return Value None

Comments Requires the Random Number Generator extension.

Example

```
'Example of Seed48(X1, X2, X3)
Seed48(bitRange1, bitRange2, bitRange3)
```

See Also SRand48

SetBackColor

SetBackColor(*BackColor*)

Sets the background color to one of the 256 colors in the color table.

Parameter	*BackColor*	Integer identifying the desired color.

Return Value None

Comments This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetBackColor(BackColor)
SetBackColor(114)
```

See Also Is35

SetBackColor16

SetBackColor16(*Red*, *Green*, *Blue*)

Sets the 16-bit background color based on the specified RGB values.

Parameters	*Red*	Integer Red color value
	Green	Integer Green color value
	Blue	Integer Blue color value

Return Value None

Comments Applies only to the Color Graphics extension. This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetBackColor16(Red, Green, Blue)
SetBackColor16(110, 47, 195)
```

See Also Is35

SetDelayToChangeEvent

SetDelayToChangeEvent(*Duration*)

Sets the delay between the last user interaction with the handheld device and the firing of the `AfterChange` event.

Parameter	*Duration*	Time in milliseconds.

Return Value None

Comments `SetDelayToChangeEvent` is a method of the `App` object. After the user enters a keystroke, the engine waits the specified duration before firing an `AfterChange` event. If the user enters another keystroke before the duration elapses, the engine waits the full specified duration before firing an `AfterChange` event.

SetFillColor

SetFillColor(*Color*)

Sets the Fill color to one of the 256 colors in the color table.

Parameter	*Color*	Integer identifying the desired color.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions. This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetFillColor(Color)
SetFillColor(255)
```

See Also Is35

SetFocus

Controls(*ControlName*).SetFocus

Puts the cursor in the specified control.

Parameter	*ControlName*	Name of a control.

Return Value None

Comments `SetFocus` is a method of the `Control` object. It applies only to the Edit and Paragraph controls.

If you are using an `OnValidate` event handler to validate user input and validation fails, you may want to use this method to place the cursor on the offending control. In the case of other controls, which do not use cursors, an alternative is to use the `Visible` property to flash the offending control a few times to attract the user's attention.

SetFocus (Bar Code Reader)

Extension.SetFocus(*str*)

Sets focus to the Bar Code Reader control after scan.

Parameter	*str*	"On" Moves cursor to the Bar Code Reader control after scan.
		"Off" Leaves cursor on the current control after scan.

Return Value None

Comments Applies only to the Bar Code Reader extension. Default="Off"

Example

```
'Example of SetFocus(str)
BarCode1.SetFocus("On")
```

SetFocus (Symbol Integrated Scanner)

Extension.SetFocus(*str*)

Sets focus to the desired control after scan.

Parameter	*str*	"On" Moves the cursor to the next edit control to be scanned.
		"Off" Moves the cursor to the Symbol Integrated Scanner control after scan.

Return Value None

Comments Applies only to the Symbol Integrated Scanner extension.

Example

```
'Example of SetFocus(str)
SIScanner1.SetFocus("On")
```

SetFont

SetFont(*font*)

Sets font style for the `DrawText` method.

Parameter *font* Integer identifying the desired font. Valid values are: 0 = stdFont, 1 = boldFont, 2 = largeFont, 3 = symbolFont, 4 = symbol1Font, 5 = symbol7Font, 6 = ledFont.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of SetFont(font)
SetFont(1)
```

See Also DrawText

SetForeColor

SetForeColor(*ForeColor*)

Sets the foreground color to one of the 256 colors in the color table.

Parameter *ForeColor* Integer identifying the desired color.

Return Value None

Comments This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetForeColor(ForeColor)
SetForeColor(255)
```

See Also Is35

SetForeColor16

SetForeColor16(*Red*, *Green*, *Blue*)

Sets the 16-bit foreground color based on the specified RGB values.

Parameters

Red	Integer Red color value.
Green	Integer Green color value.
Blue	Integer Blue color value.

Return Value None

Comments Applies only to the Color Graphics extension. This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetForeColor16(Red, Green, Blue)
SetForeColor16(255, 255, 205)
```

See Also Is35

SetIndex

Extension.SetIndex(*int*)

Bar Code Reader: Sets the index of the current control. Uses the next control for the next scan.

Symbol Integrated Scanner: Sets the index of the next edit control to scan.

Parameter *int* Index of the desired control.

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of SetIndex(int)
BarCode1.SetIndex(1)
```

SetMaxScreenDepth

SetMaxScreenDepth()

Sets the Palm screen to the maximum screen depth. Use `RestoreScreenDepth` to restore to the screen to its original depth. You can use this method to enable grayscale support on a monocrhome Palm handheld.

Parameters None

Return Value None

Comments Requires the PxScreen Tool extension.

Caution: You must call this method in the `AfterAppStart` global script. Otherwise, using a menu crashes the handheld device.

Example

```
'Example of SetMaxScreenDepth()
SetMaxScreenDepth()
```

See Also RestoreScreenDepth

SetPaintKeyCode

SetPaintKeyCode(*KeyCode*)

Specifies a virtual keycode to be sent to your app by the extension as a signal to redraw your graphics.

Parameters *KeyCode* An integer keycode value

Return Value None

Comments Applies only to the Color Graphics extension, and to the Pocket PC platform only. This function allows you to gain greater control over the painting of custom grpahics on your form, by signalling when to redraw your graphics. The extension will post this keycode when your form should be redrawn. Trap this keycode in the OnKey event, and redraw your custom graphics in response.

Example

SetPattern

SetPattern(*F0*,*F1*,*F2*,*F3*)

Sets the 8 x 8 custom Fill pattern.

Parameters	*F0*	Integer pattern value.
	F1	Integer pattern value.
	F2	Integer pattern value.
	F3	Integer pattern value.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions. Integer parameter values create an 8x8 bitmap pattern.

Example

```
'Example of SetPattern(F0,F1,F2,F3)
SetPattern(5,5,2,3)
```

SetPenColor

SetPenColor(*Color*)

Sets the Pen color to one of the 256 colors in the color table.

Parameter *Color* Integer identifying the desired color.

Return Value None

Comments Applies only to the Graphics and Color Graphics extensions.

Example

```
'Example of SetPenColor(Color)
SetPenColor(55)
```

SetPort

Extension.SetPort(*baud*, *HwHs*, *Data*, *Stop*, *Parity*)

Configures the serial port.

Parameters

baud	Sets Baud rate (i.e. 9600)
HwHs	Handshake: TRUE=CTS/RTS; FALSE = Xon/Xoff
Data	7 or 8 data bits
Stop	1 or 2
Parity	0=None; 1=Even; 2=Odd

Return Value Always TRUE for Symbol Integrated Scanner extension. None for other extensions.

Comments Applies only to the Bar Code Reader, Printer, and Symbol Integrated Scanner extensions. Default settings: `SetPort(9600, TRUE, 8, 1, 0)`

Note: The Symbol Integrated Scanner extension ignores this method.

Example

```
'Example of SetPort(baud, HwHs, Data, Stop, Parity)
BarCode1.SetPort(9600, TRUE, 7, 1, 0)
```

SetPosition

Controls(*ControlName*).SetPosition(cX, cY, cW, cH)

Modifies the current position (cX, cY) and size (cW, cH) of a control.

Parameters

ControlName	Name of a control.
cX	The new top left X coordinate of the control.
cY	The new top left Y coordinate of the control.
cW	The new width of the control.
cH	The new height of the control.

Return Value None

Comments SetPosition is a method of the Control object. This method is useful when combined with the new Dynamic Input Area support for PalmOS that enables you to move and resize controls on a form in response to changes in the form size or orientation.

Example

```
'example of control GetPosition and SetPosition methods
Dim cX, cY, cW, cH
'obtain the current location and size of Button1
Button1.GetPosition(cX, cY, cW, cH)
'move Button1 control down 10 pixels, right 10 pixels
'and widen by 5 pixels, increase height by 5 pixels
Button1.SetPosition(cX+10, cY+10, cW+5, cH+5)
```

See Also GetPosition

SetPrinter

SetPrinter(*ID*)

Only the Seiko DPU-414, Epson compatible printer is defined. No additional printers are supported. Must be set to 0.

Parameter *ID* ID of the printer to use . No additional printers are supported. Must be set to 0.

Return Value None

Comments Requires the Printer extension.

SetSelection

Controls(*ControlName*).SetSelection(*StartSel*, *EndSel*)

Highlights text in a Paragraph or Edit control.

Parameters *ControlName* Name of a control.
StartSel Offset of beginning of selection.
EndSel Offset at end of selection.

Return Value None

Comments `SetSelection` is a method of the `Control` object. Set the selection using the *StartSel* and *EndSel* parameters. *StartSel* contains the offset of the beginning of the selection – the offset of the first character in a control is 0 – and *EndSel* contains the offset to the character following the end of the selection. For example, if a control contains the string “ABCD”, to highlight the “BC” portion, call this method with *StartSel* = 1 and *EndSel* = 3.

SetTermChar

Extension.SetTermChar(chr(*x*))

Sets termination character to use.

Parameter *x* The desired termination character.

Return Value Always TRUE for Symbol Integrated Scanner extension. None for Bar Code Reader extension.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions. Default is carriage return (\n).

Note: The Symbol Integrated Scanner extension ignores this method.

Example

```
'Example of SetTermChar(chr(x))
BarCode1.SetTermChar(chr("\n"))
```

SetTextColor

SetTextColor(*TextColor*)

Sets the text color to one of the 256 colors in the color table.

Parameter *ForeColor* Integer identifying the desired color.

Return Value None

Comments Applies only to the Color Graphics extension. This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetTextColor(TextColor)
SetTextColor(2)
```

See Also Is35

SetTextColor16

SetTextColor16(*Red*, *Green*, *Blue*)

Sets the 16-bit text color based on the specified RGB values.

Parameters *Red* Integer Red color value.
Green Integer Green color value.
Blue Integer Blue color value.

Return Value None

Comments Applies only to the Color Graphics extension. This method requires Palm OS 3.5 or greater. Use `Is35` to test the handheld device OS version.

Example

```
'Example of SetTextColor16(Red, Green, Blue)
SetTextColor16(0, 0, 255)
```

See Also Is35

SetTimer

SetTimer(*Interval*)

Turns on the periodic timer and sets it with the specified period.

Parameter *Interval* The interval, in milliseconds, between timer events.

Return Value None

Comments `SetTimer` is a method of the `App` object. When you turn on the timer , the first `OnTimer` event occurs *Interval* milliseconds later. Subsequently, every *Interval* milliseconds an `OnTimer` event fires. To stop the timer, use `KillTimer`.

Use `OnTimer` for tasks that should run in the background – not interfere with user input – or to check (poll) hardware periodically.

See Also KillTimer

SetVisible

Extension.SetVisible(*boolean*)

Sets the position of the control.

Parameter *boolean* 0 = Control is invisible.

1 = Control is visible.

Return Value None

Comments Applies only to the Slider and Color Slider controls.

Example

```
'Example of SetVisible(boolean)
ColorSlider1.SetVisible(1)
```

Show

Object.Show

Makes an object visible.

Parameter *Object* Name of an object

Return Value None

Comments `Show` is a method of the Form object. Using the `Show` method of a form is equivalent to jumping to the form. The jump occurs after the script has completely executed. This means that you can reference controls in the current form even after you use `Show`, as shown in the example for this method.

Example

```
'Example of the Show method
'InputA and OutputA are edit controls in
'the current form.
Emps is another form.
'Jump to the Emps form after doing a calculation.
Forms("Emps").Show
OutputA = InputA * 10
Delay(2000)
```

See Also Visible

SIN

SIN(*x*)

Calculates the sine of the specified number.

Parameter	*x*	The number for which to calculate the sine.

Return Value The sine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of SIN(x)
Dim z
z = ASIN(x)
```

SINH

SINH(*x*)

Calculates the hyperbolic sine of the specified number.

Parameter	*x*	The number for which to calculate the hyperbolic sine.

Return Value The hyperbolic sine of the specified number.

Comments Requires the Math extension.

Example

```
'Example of SINH(x)
Dim z
z = SINH(x)
```

SkipAdvance

Extension.SkipAdvance(*t/f*)

Indicates whether to stay on same control for the next scan. For the Symbol Integrated Scanner extension, forces `Process` re-scan the last control.

Parameter	*t/f*	TRUE = Stays on same control for next scan; FALSE = Advances to the next control for the next scan.

Return Value Always returns TRUE for the Symbol Integrated Scanner control. None for Bar Code Reader control.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of SkipAdvance(t/f)
BarCode1.SkipAdvance(TRUE)
```

See Also Process

SldGetPosition

Extension.SldGetPosition()

Returns the current position of the control.

Parameters None

Return Value The current position of the control.

Comments Applies only to the Slider and Color Slider controls.

Example

```
‘Example of SldGetPosition()
Dim x
x = ColorSlider1.SldGetPosition()
```

SldSetMinMax

Extension.SldSetMinMax(*min*, *max*)

Sets the minimum and maximum values.

Parameters	*Min*	Integer minimum slider value.
	Max	Integer maximum slider value.

Return Value None

Comments Applies only to the Slider and Color Slider controls.

Example

```
'Example of SldSetMinMax(min, max)
ColorSlider1.SldSetMinMax(0, 50)
```

SldSetPosition

Extension.SldSetPosition(*n*)

Sets the position of the control.

Parameter	*n*	Desired slider position.

Return Value None

Comments Applies only to the Slider and Color Slider controls.

Example

```
'Example of SldSetPosition(n)
ColorSlider1.SldSetPosition(23)
```

SQROOT

SQROOT(*x*)

Calculates the positive square root of the specified number.

Parameter	*x*	The number for which to calculate the positive square root.

Return Value The positive square root of the specified number.

Comments Requires the Math extension.

Example

```
'Example of SQROOT(x)
Dim z
z = SQROOT(x)
```

SqrRoot

SqrRoot(*x*)

Calculates the square root of the specified number.

Parameter	*x*	The number for which to calculate the square root.

Return Value The square root of the specified number.

Comments Requires the Square Root extension.

Example

```
'Example of SqrRoot(x)
Dim z
z = SqrRoot(x)
```

SRand48

SRand48(*value*)

Assigns the highest 32 bits of `X[0]` to the specified integer value. Assigns the remaining 16 bits `0x330E`.

Parameter	*value*	The desired integer value.

Return Value None

Comments Requires the Random Number Generator extension.

Example

```
'Example of SRand48(value)
SRand48(3532)
```

See Also Seed48

Str

Str*(Variable)*

Converts an integer or floating-point number to a string.

Parameter *Variable* Integer or floating-point value to be converted to a string value.

Return Value String representation of the input value.

Comments Use the `Str` operator to force a string conversion.

Example

```
'Example of Str conversion
'InputA through InputD are edit controls
'not tied to any column.
'Try comparing the values 10 and 4, first
'as numbers, then as strings.

If Float(InputA) > Float(InputB) Then
    MsgBox("Greater Float is " & InputA)
ElseIf Float(InputB) > Float(InputA) Then
    MsgBox("Greater Float is " & InputB)
Else
    MsgBox("Floats are equal")
EndIf
If Str(InputA) > Str(InputB) Then
    MsgBox("Greater String is " & InputA)
ElseIf Str(InputB) > Str(InputA) Then
    MsgBox("Greater String is " & InputB)
Else
    MsgBox("Strings are equal")
EndIf
```

See Also Float, Int, Int64

StrCompare

StrCompare(*string1, string2*)

Performs a case sensitive comparison of two strings.

Parameter *String1* The first string to compare.

String2 The second string to compare.

Return Value Returns True if the strings match exactly, False if not.

Comments Requires the Strings extension.

Example

```
'Example of StrCompare
Dim match
match = StrCompare(“ABCD”, “abCD”)
'Result: match is False
```

See Also StrCompSort

StrCompSort

StrCompSort(*string1, string2*)

Performs a case sensitive comparison of two strings.

Parameter *String1* The first string to compare.

String2 The second string to compare.

Return Value Returns 0 if the strings match exactly, returns a positive number if String1 sorts after String2 alphabetically, returns a negative number if String1 sorts before String2 alphabetically.

Comments Requires the Strings extension.

Example

```
'Example of StrCompSort
Dim match
match = StrCompSort("ABCD", "abCD")
'Result: match contains 1
```

See Also StrCompare

String

String*(length, character)*

Returns a String containing a repeating character string of the length specified.

Parameter *Length* Length of desired string of repeating characters.

Character Character to repeat in returned string.

Return Value String of repeating characters of desired length.

Comments Requires the Strings extension.

Example

```
'Example of String function
MsgBox(String(5, "G"))
'Displays MsgBox containing "GGGGG"
```

StripTerm

Extension.StripTerm(*t/f*)

Indicates whether to strip the termination character from the end of the string `GetScan` returns.

Parameter *t/f* TRUE = Strips termination character; FALSE = Leaves the termination character in place.

Return Value None

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions. Default = TRUE.

Note: The Symbol Integrated Scanner extension ignores this method.

Example

```
'Example of StripTerm(t/f)
BarCode1.StripTerm(TRUE)
```

See Also GetScan

SU_BlockAllHotKeys

SU_BlockAllHotKeys(*enable*)

Blocks all hotkey keypresses, but does not send a keypress that can be caught in the OnKey event.

Parameter *enable* True to enable, False to disable

Return Value None

Comments Requires the SysUtils extension.

Example

```
SU_BlockAllHotKeys(true)
```

See Also

SU_CheckSystemPassword

SU_CheckSystemPassword(*password*)

Verifies if the passed string is the current system password or not.

Parameter *password* String containing password to verify.

Return Value Returns 1 (True) if password matches, or 0 (False) if it does not match.

Comments Requires the SysUtils extension.

Example

```
isPassword = SU_CheckSystemPassword(“password”)
```

See Also

SU_ClipboardTextGet

SU_ClipboardTextGet()

Get the current text clip from the system clipboard.

Parameter *None*

Return Value Returns a string containing the current system clipboard text.

Comments Requires the SysUtils extension.

Example

```
cliptext = SU_ClipboardTextGet()
```

See Also

SU_ClipboardTextSet

SU_ClipboardTextSet(*cliptext*)

Set the system clipboard text clip to the passed string.

Parameter *cliptext* String of text to set the clipboard with.

Return Value None

Comments Requires the SysUtils extension.

Example

```
SU_ClipboardTextSet(“important text here”)
```

See Also

SU_DelAppPref

SU_DelAppPref(*CreatorID, prefindex*)

Deletes the saved application preference setting.

Parameter *CreatorID* 4-character case-sensitive application Creator ID string.

prefindex Positive 16-bit integer specifying preference “slot” or “row”.

Return Value None

Comments Palm OS only. Requires the SysUtils extension. Palm OS application preferences are stored in the Saved Preferences system database, referenced by Creator ID and index. Think of the prefindex as a “slot” or “row” number. Think of a preference setting as an equivalent to a Windows registry entry. Preferences can be used to store persistent information outside of your data tables. For an equivalent function on the Pocket PC, see SU_RegDeleteKey.

Example

```
SU_DelAppPref(“MYAP”, 50)
```

See Also

SU_GetAppPref

SU_GetAppPref(*CreatorID, prefindex*)

Returns the saved preference value for the given Creator ID and prefindex.

Parameter	*CreatorID*	4-character case-sensitive application Creator ID string.
	prefindex	Positive 16-bit integer specifying preference “slot” or “row”.

Return Value String of text stored in the specified Saved Preferences “slot”.

Comments Palm OS only. Requires the SysUtils extension. Palm OS application preferences are stored in the Saved Preferences system database, referenced by Creator ID and index. Think of the prefindex as a “slot” or “row” number. Think of a preference setting as an equivalent to a Windows registry entry. Preferences can be used to store persistent information outside of your data tables. For an equivalent function on the Pocket PC, see SU_RegReadKey.

Example

```
preftext = SU_GetAppPref(“MYAP”, 50)
```

See Also

SU_GetBatteryPercent

SU_GetBatteryPercent()

Returns current battery charge level as a percentage (0-100) of full.

Parameter *None*

Return Value Current battery charge level as a percentage (0-100) of full.

Comments Requires the SysUtils extension.

Example

```
battlevel = SU_GetBatteryPercent()
```

See Also

SU_GetDeviceID

SU_GetDeviceID()

Returns the device unique ID string.

Parameter *None*

Return Value String containing the device unique ID.

Comments Requires the SysUtils extension. Not all Palm OS devices have a unique ID string. For PalmOS devices that do not have a unique device ID, the string "No FlashID" is returned, otherwise the device's 12 character alphanumeric unique ID is returned. For Pocket PC 2002/2003 devices, the device ID is usually 13 characters long, but for WM5/WM6 devices it is much longer at 40 characters.

Example

```
devID = SU_GetDeviceID()
```

See Also

SU_GetDeviceModel

SU_GetDeviceModel()

Returns the device model string to help identify the device.

Parameter *None*

Return Value String containing the device model identifier.

Comments Requires the SysUtils extension. For PalmOS devices, this returns the case sensitive 4-character CompanyID (eg. “Palm”) plus the 4-character Device Model ID (eg. “D060”). Some very old PalmOS devices may not have a Model ID. For PocketPC devices this returns the manufacturer's device model identification string (eg. “Symbol MC50”).

Example

```
devModel = SU_GetDeviceModel()
```

See Also

SU_GetMemInfo

SU_GetMemInfo(*memorytype*)

Returns the amount of available memory on the device.

Parameter *memorytype* Pass a memorytype value of 0 for Free RAM, or 1 for RAM size, or 2 for ROM size.

Return Value Available memory in kilobytes (KB).

Comments Requires the SysUtils extension. For Pocket PC, the ROM size reported is actually the RAM size (specifying RAM or ROM returns the same value).

Example

```
'Get the amount of free RAM available
membytes = SU_GetMemInfo(0)
```

See Also

SU_GetOSVersion

SU_GetOSVersion()

Returns the device operating system version string.

Parameter *None*

Return Value String containing the device OS version.

Comments Requires the SysUtils extension.

Example

```
OSver = SU_GetOSVersion()
```

See Also

SU_GetOwnerName

SU_GetOwnerName()

Returns the device owner name.

Parameter *None*

Return Value String containing the device owner name.

Comments Requires the SysUtils extension. For Palm OS devices, this returns the HotSync user name, the same as the GetUserName script function. For Pocket PC devices, this owner name is different than the value returned by the GetUserName script.

Example

```
owner = SU_GetOwnerName()
```

See Also

SU_GetPlatform

SU_GetPlatform()

Returns the device OS platform string, either PALMOS or POCKETPC.

Parameter *None*

Return Value Device OS platform string, either PALMOS or POCKETPC.

Comments Requires the SysUtils extension. Use this function to distinguish if the current device runs on the Palm OS or Pocket PC platform.

Example

```
strplatform = SU_GetPlatform()
```

See Also

SU_GetPluggedIn

SU_GetPluggedIn()

Returns whether or not the device is currently plugged in to AC power.

Parameter *None*

Return Value Returns 1 (true) if AC power is currently connected or 0 (false) if not.

Comments Requires the SysUtils extension.

Example

```
pluggedin = SU_GetPluggedIn()
```

See Also

SU_HideStartIcon

SU_HideStartIcon(*enable*)

Hides the Start icon in the top left of the Windows Mobile screen.

Parameter *enable* True to enable, False to disable

Return Value None

Comments Requires the SysUtils extension. Windows Mobile platform only. Hides or shows the Start icon in the top left corner of the screen. This function may not be effective on all devices, as the device manufacturer may have modified the OS API that controls this function.

Example

```
'hide the start icon
SU_HideStartIcon(true)
```

See Also

SU_HotSync

SU_HotSync()

Initiates a standard cradle/cable HotSync.

Parameter *None*

Return Value None

Comments Requires the SysUtils extension. Palm OS platform only. Initiates HotSync by enqueuing HotSync virtual keypress (decimal 521).

Example

```
'start cradle HotSync
SU_HotSync()
```

See Also

SU_LaunchApp

SU_LaunchApp(*strFilename, strParam*)

Launch a specified application, and pass an optional parameter.

Parameter *strFilename* File name of the application to launch.

strParam Optional parameter to pass to launched application.

Return Value Returns 0 if the application launched okay, or an error code otherwise.

Comments Requires the SysUtils extension. For the Palm OS platform, the application name is case sensitive and must not include any path (only applications in internal memory can be launched). If you specify a parameter string, it is passed to the launching app as the command parameter block. For the Pocket PC platform, specify the full path and name of the application, or document, or URL to launch. If you specify a parameter string, it is passed as a commandline parameter when launching the application.

Example

```
'launch calculator
If SU_GetPlatform() = “PALMOS” then
    SU_LaunchApp(“Other App”, “”)
Else
    SU_LaunchApp(“\Program Files\Other App\Other App.exe”, “”)
EndIf
```

See Also

SU_LaunchAppAtEvent

SU_LaunchAppAtEvent(*strFilename, event*)

Launch a specified application/document/URL at a specified system event.

Parameter *strFilename* Full path and file name of the application to launch.

event Numeric system event code for 0-12, per event table below.

Return Value Returns 0 if the application launched okay, or an error code otherwise.

Comments Requires the SysUtils extension. Pocket PC platform only. Specify the full path and name of the application, or document, or URL to launch when the specified system event occurs. Supported Pocket PC system events include:

Event Code	Event Description
0	EVENT_NONE - used to cancel an existing event
1	EVENT_TIME_CHANGE
2	EVENT_SYNC_END
3..6	*not supported by Pocket PC OS*
7	EVENT_DEVICE_CHANGE
8	*not supported by Pocket PC OS*
9	EVENT_RS232_DETECTED
10	EVENT_RESTORE_END
11	EVENT_WAKEUP
12	EVENT_TZ_CHANGE

Example

```
'launch calculator when device wakes up (event code 11)
SU_LaunchAppAtEvent(“\Windows\calc.exe”, 11)
```

See Also

SU_LaunchAppAtTime

SU_LaunchAppAtTime(*strFilename, systemdate*)

Launch a specified application/document/URL at a specified date and time.

Parameter *strFilename* Full path and file name of the application to launch.

systemdate Date and time in SatForms system date format.

Return Value Returns 0 if the application launched okay, or an error code otherwise.

Comments Requires the SysUtils extension. Pocket PC platform only. Specify the full path and name of the application, or document, or URL to launch at the specified date and time. Specify the date and time using SatForms system date format, which is the number of seconds since 00:00:00 Jan 1, 1904, encompassing both a date and time into a single value.

Example

```
'launch calculator at 4:15 PM today
'there are 86400 seconds per day
dim seconds
seconds = (GetSysDate * 86400) + TimeToSysTime("4:15pm")
SU_LaunchAppAtTime("\Windows\calc.exe", seconds)
```

See Also

SU_ModemHotSync

SU_ModemHotSync()

Initiates standard modem/network hotsync.

Parameter *None*

Return Value None

Comments Requires the SysUtils extension. Palm OS platform only. Initiates modem or network HotSync by enqueuing modem HotSync virtual keypress (decimal 522).

Example

```
'start modem HotSync
SU_ModemHotSync()
```

See Also

SU_ParseDelimText

SU_ParseDelimText(*string, chunkpos, delimiter*)

Returns a chunk of data in a string of delimited items.

Parameter	*string*	Text string of delimited items.
	chunkpos	Item position in delimited string (0-based).
	delimiter	Character used to delimit items.

Return Value String containing desired item from delimited string.

Comments Requires the SysUtils extension.

Example

```
dim item
item = SU_ParseDelimText( "ABC#DEF#GHIJ#KLMN#OPQ", 2, "#" )
'item contains "GHIJ"
```

See Also

SU_PasteChars

SU_PasteChars(*string*)

Paste a string to the keyboard input queue as though it was typed in.

Parameter *string* String of text.

Return Value None

Comments Requires the SysUtils extension. Input goes to the control that has the focus. Use this function for regular printable characters, and use SU_QueueVirtualKey for virtual keys.

Example

```
SU_PasteChars("Hello World")
```

See Also

SU_PlaySoundFile

SU_PlaySoundFile(*filename, waituntildone*)

Plays a WAV audio file on Windows Mobile devices.

Parameter *filename* Path and name of WAV file to play.

	waituntildone	True or False to wait until the sounds is done playing before returning.

Return Value None

Comments Requires the SysUtils extension. Windows Mobile platform only. Use this method to play WAV audio files.

Example

```
SU_PlaySoundFile(GetAppPath & "soundfile.WAV", false)
```

See Also

SU_PowerOff

SU_PowerOff()

Power off the device now, as if the power button had been pressed.

Parameter *None*

Return Value None

Comments Requires the SysUtils extension. On the Palm OS platform, this function works by enqueuing a power button keypress (decimal 520).

Example

```
'power off the device right now
SU_PowerOff()
```

See Also

SU_QueueVirtualKey

SU_QueueVirtualKey(*keycode*)

Post a virtual key to the keyboard input queue.

Parameter	*keycode*	Virtual character keycode.

Return Value None

Comments Requires the SysUtils extension. Input goes to the control that has the focus. Use this function for virtual keys; for regular printable characters use SU_PasteChars.

Example

```
'example to post a backspace keypress into Pocket PC key queue
SU_QueueVirtualKey(8)
```

See Also

SU_RegDeleteKey

SU_RegDeleteKey(*hive*, *key*)

Delete specified key and all settings within it from the registry.

Parameter	*hive*	String containing base registry hive.
	key	String indicating key to delete (including all settings & subkeys)

Return Value Returns 0 if no error, or error code otherwise.

Comments Requires the SysUtils extension. Pocket PC only. The base registry hive can be one of these case sensitive strings: "HKEY_LOCAL_MACHINE", "HKEY_CURRENT_USER", "HKEY_CLASSES_ROOT", or "HKEY_USERS". The Key must NOT begin with a slash character. For Palm OS equivalent use SU_DelAppPref.

Example

```
'delete "HKEY_LOCAL_MACHINE\Software\MyCompany\MyApp" key
err = SU_RegDeleteKey("HKEY_LOCAL_MACHINE",
"Software\MyCompany\MyApp")
```

See Also

SU_RegReadKey

SU_RegReadKey(*hive, keytype, key, setting*)

Read specified key setting value from the registry.

Parameter *hive* String containing base registry hive.

keytype 0 for REG_SZ strings, or 1 for REG_DWORD numeric values.

key String indicating subkey to read from.

setting String containing specific setting within the subkey to read.

Return Value Returns the value stored in that specified key and setting.

Comments Requires the SysUtils extension. Pocket PC only. The base registry hive can be one of these case sensitive strings: "HKEY_LOCAL_MACHINE", "HKEY_CURRENT_USER", "HKEY_CLASSES_ROOT", or "HKEY_USERS". The Key and Setting must NOT begin with a slash character.

Example

```
'read the HKEY_LOCAL_MACHINE\Software\Microsoft\Windows CE
Services\FileSyncPath value
value = SU_RegReadKey("HKEY_LOCAL_MACHINE", 0,
"Software\Microsoft\Windows CE Services", "FileSyncPath")
```

See Also

SU_RegWriteKey

SU_RegWriteKey(*hive, keytype, key, setting, value*)

Write specified key setting value to the registry.

Parameter *hive* String containing base registry hive.

keytype 0 for REG_SZ strings, or 1 for REG_DWORD numeric values.

key String indicating subkey to write to.

setting String containing specific setting within the subkey to write to.

value String value to store in specified key setting.

Return Value Returns 0 if no error, or error code otherwise.

Comments Requires the SysUtils extension. Pocket PC only. The base registry hive can be one of these case sensitive strings: "HKEY_LOCAL_MACHINE", "HKEY_CURRENT_USER", "HKEY_CLASSES_ROOT", or "HKEY_USERS". The Key and Setting must NOT begin with a slash character.

Example

```
'write user name string to
HKEY_LOCAL_MACHINE\Software\MyCompany\MyApp\UserName
err = SU_RegWriteKey("HKEY_LOCAL_MACHINE", 0,
"Software\MyCompany\MyApp", "UserName", "Joe User")
```

See Also

SU_Reset

SU_Reset()

Soft reset the device now.

Parameter *None*

Return Value None

Comments Requires the SysUtils extension. Performs a soft reset as if the reset pin had been depressed.

Example

```
'soft reset the device right now
SU_Reset()
```

See Also

SU_SetAppPref

SU_SetAppPref(*creatorIDstring, prefindex, prefval*)

Saves the preference value for the supplied creatorID and pref index.

Parameter *CreatorID* 4-character case-sensitive application Creator ID string.

prefindex Positive 16-bit integer specifying preference “slot” or “row”.

prefval String value to write into specified preference slot.

Return Value None

Comments Palm OS only. Requires the SysUtils extension. Palm OS application preferences are stored in the Saved Preferences system database, referenced by Creator ID and index. Think of the prefindex as a “slot” or “row” number. Think of a preference setting as an equivalent to a Windows registry entry. Preferences can be used to store persistent information outside of your data tables. For an equivalent function on the Pocket PC, see SU_RegWriteKey.

Example

```
'store value in prefindex 50 for creatorID "MYAP"
SU_SetAppPref("MYAP", 50, "Joe User")
```

See Also

SU_SetAutoOffTime

SU_SetAutoOffTime(*duration*)

Sets the auto-off timer on the Palm, in seconds.

Parameter *duration* Duration in seconds to set the auto-off timer to.

Return Value The previous auto-off timer value is returned.

Comments Requires the SysUtils extension. Palm OS only. A value of 0 indicates 'never turn off automatically'.

Example

```
'set auto-off timer to 30 seconds
SU_SetAutoOffTime(30)
```

See Also

SU_SetDeviceDateTime

SU_SetDeviceDateTime(*seconds*)

Set the current device date and time.

Parameter *seconds* Current date andtime in SatForms system date format.

Return Value None

Comments Requires the SysUtils extension. Specify the date and time using SatForms system date format, which is the number of seconds since 00:00:00 Jan 1, 1904, encompassing both a date and time into a single value.

Example

```
'set system date and time to values in edDate and edTime controls
'there are 86400 seconds per day
dim seconds
seconds = (DateToSysDate(edDate)*86400) + TimeToSysTime(edTime)
SU_SetDeviceDateTime(seconds)
```

See Also

SU_SetHotKey

SU_SetHotKey(*keycode, enable*)

Trap a hotkey keypress so that it sends a keypress that can be caught in the OnKey event.

Parameter *keycode* Keycode for the virtual key keypress you want to trap.

enable True to enable or False to disable.

Return Value None

Comments Requires the SysUtils extension. Pocket PC only. Normally, the Pocket PC OS processes hotkeys before they get to your application. When this function is enabled for a specific hotkey(s), the hotkey keypress will generate a key event that you can trap and handle in the OnKey event. The hotkey virtual keycode will be passed to the OnKey event in both the ASCII and VirtKey variables of the GetLastKey function.

Example

```
'trap hotkey VK_APP1 (decimal 193)
SU_SetHotKey(193, true)
```

See Also

SU_SysIdleTimerReset

SU_SysIdleTimerReset()

Reset the system idle timer to prevent the device from dozing off into sleep mode.

Parameter *None*

Return Value None

Comments Requires the SysUtils extension.

Example

```
SU_SysIdleTimerReset()
```

See Also

SU_TapScreen

SU_TapScreen(*x, y*)

Enqueue a virtual tap on the screen at the given X, Y position.

Parameter *X* The horizontal (X) screen coordinate of the tap.

Y The vertical (Y) screen coordinate of the tap.

Return Value None

Comments Requires the SysUtils extension. Note that on the Palm OS platform, the virtual tap on some controls may not have the same effect as an actual tap.

Example

```
SU_TapScreen(10, 20)
```

See Also

Sub

Sub *name* [(*arglist*)]

Defines a global script subroutine that can take optional parameters and perform your user-defined statements. It does not returns a result. Global functions are available to all forms and scripts in your application.

Parameters *optional arguments*

Return Value Value defined by the subroutine.

Comments Applies to the entire application. In the Global script section of the application property explorer window, there are two Global Funcs & Subs sections, labeled (shared) and (private). The (shared)section allows you to write global scripts that are shared between all platform targets in the application, for example a Palm target and a Pocket PC target.The (private) section allows you to write global scripts that exist in the current target only, so that you may have script that apply to the current target platform only. This enables you to have common code between targets in the (shared) section, and platform-specific code in the (private) section. One common use for this capability is to set a global variable to a specific value that indicates which platform your application is running on, thus enabling formlevel scripts to take appropriate action based on the current platform target.

Example

```
Sub name [(arglist)]
[statements]
[exit]
[statements]
End Sub
```

Remarks

- You can define local variables using the Dim keyword. The value of local variables in a Sub is not preserved between calls to the routine.
- You cannot define a nested subroutine within a Sub or a Function.
- You can call a subroutine using its name and its variable. Unlike Visual Basic, you do not need to use the keyword Call.
- Unlike Visual Basic, the Exit keyword cannot be fully qualified. In Visual Basic, you have to use Exit Sub to exit the routine.
- Functions and subroutines can call themselves repeatedly (recursive). Do so cautiously because excessive recursion can lead to stack overflow.

See Also Function

Sum

Tables(*TableName*).Sum(*ColumnName*)

Returns the sum of values in a specified column for all records in a table.

Parameters *TableName* Name of a table.

ColumnName Name of a numeric column in the table.

Return Value Sum of the values of the specified column for all records in the table.

Comments `Sum` is a method of the `Table` object. If a filter is in place that limits the table to certain records, only those records are used to compute the sum.

Example

```
'Example of Sum method
'OutputA is an edit control.
OutputA = Tables("Emps").Sum("Salary")
```

See Also Count

SysDateToDate

SysDateToDate(*NumDays*)

Converts days since January 1, 1904, to a user-readable date.

Parameter *NumDays* Number of days since January 1, 1904.

Return Value User-readable date in format specified in the handheld device preferences.

Comments `SysDateToDate` is a method of the `App` object.

See Also DateToSysDate, GetSysDate, GetSysTime, SysTimeToTime, TimeToSysTime

SystemDateFormat

SystemDateFormat

Returns the system date format string.

Parameter *None*

Return Value String describing the date format specified in the handheld device preferences.

Comments SystemDateFormat requires the Strings extension.

See Also DateToSysDate, FormatDate, FormatDateN, FormatTime, FormatTimeN, GetSysDate, GetSysTime, SystemTimeFormat, SysTimeToTime, TimeToSysTime

SystemTimeFormat

SystemTimeFormat

Returns the system time format string.

Parameter *None*

Return Value String describing the time format specified in the handheld device preferences.

Comments SystemTimeFormat requires the Strings extension.

See Also DateToSysDate, FormatDate, FormatDateN, FormatTime, FormatTimeN, GetSysDate, GetSysTime, SystemDateFormat, SysTimeToTime, TimeToSysTime

SysTimeToTime

SysTimeToTime(*NumSeconds*)

Converts seconds since midnight to a user-readable time.

Parameter *NumSeconds* Number of seconds since midnight

Return Value User-readable time in format specified in the handheld device preferences.

Comments `SysTimeToTime` is a method of the `App` object.

This method performs a `Mod` operation (`Mod` of 1 day), so it can be used with `GetSysTime` to find the current time.

See Also DateToSysDate, GetSysDate, GetSysTime, SysDateToDate, TimeToSysTime

TAN

TAN(*x*)

Calculates the tangent of the specified number.

Parameter *x* The number for which to calculate the tangent.

Return Value The tangent of the specified number.

Comments Requires the Math extension.

Example

```
'Example of TAN(x)
Dim z
z = TAN(x)
```

TANH

TANH(*x*)

Calculates the hyperbolic tangent of the specified number.

Parameter *x* The number for which to calculate the hyperbolic tangent.

Return Value The hyperbolic tangent of the specified number.

Comments Requires the Math extension.

Example

```
'Example of TANH(x)
Dim z
z = TANH(x)
```

TermRecd

Extension.TermRecd()

Returns TRUE if the termination character is last character in the string. Use after `GetScan` to verify that the termination character was received.

Parameters None

Return Value TRUE if the last scan succeeded; FALSE if it did not.

Comments Applies only to the Bar Code Reader and Symbol Integrated Scanner extensions.

Example

```
'Example of TermRecd()
Dim x
x = BarCode1.TermRecd()
```

See Also GetScan

TimeToSysTime

TimeToSysTime(*Time*)

Converts a user-readable time to seconds since midnight.

Parameter *Time* Time in format specified in the handheld device preferences.

Return Value Seconds since midnight.

Comments `TimeToSysTime` is a method of the `App` object.

See Also DateToSysDate, GetSysDate, GetSysTime, SysDateToDate, SysTimeToTime

Tone

Tone(*Frequency, Duration, Amplitude*)

Issues a tone of a specified frequency, duration, and amplitude.

Parameters
Frequency Frequency of the tone in Hertz.
Duration Duration of tone in milliseconds.
Amplitude Volume; range is 0–64. This parameter is optional. If it is omitted, the sound plays at the default level.

Return Value None

Example

```
'Example of Tone
'InputA and InputB are edit controls.
Tone(InputA, InputB)
```

See Also Beep

Trim

Trim(*string*)

Trims leading and trailing spaces from input string.

Parameters *String* The string to trim leading and trailing spaces from.

Return Value The input string trimmed of leading and trailing spaces.

Comments Requires the Strings extension.

Example

```
'Example of Trim
Dim strTest
strTest = Trim("   ABCD   ")
'Result: strTest contains "ABCD"
```

See Also LTrim, RTrim

UCase

UCase(*string*)

Converts string to all upper case.

Parameter *string* The string to convert to all upper case.

Return Value The input string converted to all upper case.

Comments Requires the Strings extension.

Example

```
'Example of UCase
Dim strTest
strTest = UCase("string to convert")
'returns "STRING TO CONVERT"
```

See Also LCase

Underline

Controls(*ControlName*).Underline

Returns or sets the underline attribute of a control.

Parameter *ControlName* Name of a control.

Return Value None

Comments Underline is a property of the `Control` object. This property is only available for Edit and Paragraph controls.

Version

Version()

Returns the version number of this extension.

Parameters None

Return Value The version number of this extension.

Comments Requires the PxScreenTool extension.

Example

```
'Example of Version()
Dim x
x = Version()
```

Visible

Controls(*ControlName*).Visible

Indicates whether a control is visible. Assigning a value to the `Visible` property makes a control visible or invisible.

Parameter *ControlName* The name of a control.

Return Value TRUE for visible controls; FALSE for invisible controls

Comments `Visible` is a property of the `Control` object. Use it to make a control visible or invisible or to determine if a control is visible or invisible.

Multiple controls can occupy the same coordinates on the screen. Using the Visible property, you can make different controls visible to the user at different times.

Example

```
'Example of Visible property
'InputA, InputB, and OutputA through
'OutputD are edit controls.
'Access the visible property of a control.
OutputA = InputA.Visible
OutputB = InputB.Visible
'Make a control visible.
OutputC.Visible = TRUE
'Make a control invisible.
OutputD.Visible = FALSE
```

See Also Show

While … Wend

While *Condition*
Statements
[Exit While]
Wend

Perform a while loop.

Parameters *Condition* Condition to evaluate.

Statements Commands to execute if *Condition* evaluates to TRUE.

Return Value None

Comments The `Exit While` clause is optional and is used to exit the loop prematurely. The loop exits after the `Wend` statement. If there is no `Exit While` clause, the loop continues to execute until *Condition* evaluates to FALSE.

Caution: If *Condition* never evaluates to FALSE, the loop runs infinitely.

Example

```
'Example of While loop
Dim x
x = 1
While x < 10
    MsgBox("The value of x is " & x)
    If x = 5 Then Exit While
    x = x + 1
Wend
MsgBox ("Done.")
```

See Also For…To…Next

X_Buffer

X_Buffer(*value*)

Returns the current `X[i]` value from `X[i] = (a*X[i-1] + c)mod` 2^{48}.

Parameter *value* Must be 1, 2 or 3, designating the desired 16-bit order.

Return Value The current `X[i]` value from `X[i] = (a*X[i-1] + c)mod` 2^{48}.

Comments Requires the Random Number Generator extension.

Example

```
'Example of X_Buffer(value)
Dim x
x = X_Buffer(1)
```

X_Initial

X_Initial(*value*)

Returns 16 bits of the initial 48-bit `X[i]` value.

Parameter *value* Must be 1, 2 or 3, designating the desired 16-bit order.

Return Value Sixteen bits of the initial 48-bit `X[i]` value.

Comments Requires the Random Number Generator extension.

Example

```
'Example of X_Initial(value)
Dim x
x = X_Initial(3)
```

X_Last

X_Last(*value*)

Returns the last `X[i-1]` value from `X[i] = (a*X[i-1] + c)mod` 2^{48}.

Parameter *value* Must be 1, 2 or 3, designating the desired 16-bit order.

Return Value The last `X[i-1]` value from `X[i] = (a*X[i-1] + c)mod` 2^{48}.

Comments Requires the Random Number Generator extension.

Example

```
'Example of X_Last(value)
Dim x
x = X_Last(1)
```

Xor [bitwise]

Number1* Xor *Number2

Performs a bitwise `Xor` operation between the operands.

Parameters N*umber1* First operand.

Number2 Second operand.

Return Value The result of a bitwise `Xor` of the two operands.

Example

```
'Example of bitwise Xor
'InputA is an edit control.
'Toggle bit 4 (mask = 10 hex) in number.
OutputA = InputA Xor &H10
```

See Also And [bitwise]And [logical], Or [bitwise], Not [bitwise]

Xor [logical]

Condition1* Xor *Condition2

Joins two conditions where one and only one condition must evaluate to TRUE for the statement to evaluate to TRUE.

Parameters *Condition1* First condition to be evaluated.

Condition2 Second condition to be evaluated.

Return Value TRUE if *Condition1* evaluates to TRUE and *Condition2* evaluates to FALSE; TRUE if *Condition1* evaluates to FALSE and *Condition2* evaluates to TRUE; FALSE otherwise.

Comments Each condition is evaluated individually, then the `Xor` operation is performed. Note that `Xor`, `And`, `Or`, and `Not` only perform Boolean operations if both conditions are Boolean. Otherwise, they perform bitwise operations on their operands.

See Also And [bitwise]And [logical], Or [logical], Not [logical]

Chapter 12
Satellite Forms API Reference

This chapter provides instructions on how to use the Satellite Forms Application Programming Interface (API) to write extensions and modify extensions written by others. It includes a complete Satellite Forms API reference.

Satellite Forms API Overview

The Satellite Forms API enables you to extend the capabilities of Satellite Forms by writing extensions, called SFX plug-ins and SFX Custom controls, or by modifying extensions written by others. Satellite Forms extensions are programs written in C that exchange data with the Satellite Forms Engine on handheld devices. You can use extensions to manipulate data, perform complex business logic, create custom controls, pop up dialog boxes, and handle many other functions. For example, with the Satellite Forms API, you can do any of the following:

- Extend the Satellite Forms scripting language with libraries of new functions, including financial, statistical, and so on.
- Create new user-interface objects: Satellite Forms SFX Custom controls.
- Build drivers for devices such as barcode scanners, portable printers, or pagers.
- Add business logic and complex validation to a form.

C programs that use the Satellite Forms API to interact with the Satellite Forms engine are called Satellite Forms extensions. There are two kinds of extensions:

- SFX plug-ins
- SFX Custom controls

Satellite Forms SFX plug-ins are C-language functions called from a script in a Satellite Forms application at run time, perform some kind of logic or data manipulation, and optionally return data to the Satellite Forms application.

Satellite Forms SFX Custom controls are new custom controls that appear in the MobileApp Designer at design time and can be incorporated into an application just like one of the built-in Satellite Forms controls.

When writing your own extensions, be sure to check the version of the Satellite Forms engine to ensure compatibility with your extension code. See SF_GetEngineVersion on page 500 for a description of the applicable API function.

Satellite Forms comes with several sample extensions, including an SFX plug-in called Square Root and an SFX Custom control called Slider. If you installed Satellite Forms in the default installation directory, the source code for the Generic and Square Root extensions and the Slider SFX Custom control extension is located in:
C:\Satellite Forms 8\Samples\Extensions\Generic\Src\
C:\Satellite Forms 8\Samples\Extensions\Square Root\Src\
C:\Satellite Forms 8\Samples\Extensions\Slider\Src\

If you installed Satellite Forms in the default installation directory, the executable code for these and other extensions for Palm and Pocket PC is located in:
C:\Satellite Forms 8\Extensions\Standard\Palm\
C:\Satellite Forms 8\Extensions\Standard\PPC_ARM\

The next section presents examples of using the API to create both an SFX plug-in and an SFX Custom control.

Creating an SFX plug-In

While the Satellite Forms scripting language currently supports basic mathematical operations, some applications may require additional capabilities. Using the Satellite Forms API, you can create extensions that provide any needed mathematical functions. This example uses the Square Root extension, which returns the square root of a number.

The following example illustrates how to create the Square Root SFX plug-in extension supplied with Satellite Forms. You should be familiar with loading and using extensions in Satellite Forms applications.

To create an SFX plug-in extension, you must complete the following steps, described in detail below:

1. Create a development directory with all necessary files.
2. Write the C-language extension.
3. Create a descriptor file for the extension.
4. Copy the extension and descriptor file to a subdirectory of the Satellite Forms Extensions directory.
5. Load the extension into a Satellite Forms application.
6. Download the extension and test application to the handheld device.
7. Test the extension by calling it from a Satellite Forms script.

Step 1: Create a development directory with all necessary files

The easiest way to begin writing an extension is to copy the contents of one of the sample extension directories to your development directory for the new extension. Create the development directory for the extension and, since this is an SFX plug-in project, copy to it the contents of either the Generic or Square Root sample extension directories. If you installed Satellite Forms in the default installation directory, the source code for the Generic and Square Root extensions is located in:
C:\Satellite Forms 8\Samples\Extensions\Generic\Src\
C:\Satellite Forms 8\Samples\Extensions\Square Root\Src\

Step 2: Write and compile the C-language extension

You can write the C-language extension using Metrowerks' CodeWarrior for PalmOS, that allows you compile the code to a .PRC file for Palm OS, or Microsoft Embedded Visual C++ to compile an .SFX file for Pocket PC OS. If you use CodeWarrior, you can also use the Metrowerks debugger with your Satellite Forms extensions.

Extensions are event handlers. You begin an extension by declaring a structure for the necessary global variables. You then specify the events that you want your extension to handle. Extensions can handle any of the events of the scripting language as well as some additional events that are specific to extensions. The available event types are defined in the SFDefs.h file and are listed and described in the following tables. If you installed Satellite Forms in the default installation directory, the SFDefs.h file is located in:
C:\Satellite Forms 8\Include\

The following table lists and describes the standard events available to extensions.

Note The `SF_Handler_Form_AfterOpen` and `SF_Handler_Form_BeforeClose` events in the API differ from the corresponding events in the scripting language. In the scripting language, these events fire one time per form, upon entering and exiting the form. In the Satellite Forms API, these events fire one time per page, upon entering and exiting the page.

Table 12.1 Standard events available to extensions

Event	Description
SF_Handler_Form_AfterOpen	Occurs after a form or a form's page is opened.
SF_Handler_Form_Afterload	Occurs after the controls on a form are loaded with data from the form's table.
SF_Handler_Form_AfterChange	Occurs after data in any field of the form is changed.
SF_Handler_Form_AfterRecCreate	Occurs after a new record is created in a form's linked table.
SF_Handler_Form_BeforeClose	Occurs before a form or page is closed (exited).
SF_Handler_Form_BeforeRecDelete	Occurs before a record is deleted from a form (can prevent delete).
SF_Handler_Form_OnValidate	Occurs when a form is validated (can fail validation).

The following table lists and describes extension-specific events:

Table 12.2 Extension-specific events

Event	Description
SF_Handler_OnExtLoad	Occurs when the application is opened and the extension is loaded.
SF_Handler_OnExtUnload	Occurs when the application is exited and the extension is unloaded. Use the `OnExtUnload` event to free any allocated globals.
SF_Handler_OnFormEvent	Occurs when an OS event reaches a form. You can handle the event in your extension or allow the OS to handle it.
SF_Handler_OnSysEvent	Occurs with every OS event before the OS receives event notification. You can handle the event in your extension or allow the OS to handle it.
SF_Handler_OnSFXNotify	Occurs when a user accesses one of the methods of an SFX control or the OS notifies an SFX control of certain actions.

When you are finished writing the code, compile the extension for the desired platform.

Step3: Create a descriptor file for the extension

The descriptor file includes information about the extension and describes its methods and parameters. This information appears in MobileApp Designer and enables other developers to use your extension properly in their Satellite Forms applications. Descriptor files have the .INF extension. Refer to the examples in each of the extension sample subdirectories for more information. If you installed Satellite Forms in the default installation directory, the .INF files are located in: C:\Satellite Forms 8\Samples\Extensions\<*extension_name*>\

Step 4: Copy the extension and descriptor file to a subdirectory of the Satellite Forms Extensions directory

After you write and compile the Satellite Forms extension and create its descriptor file, copy the extension files to the following subdirectories of your Satellite Forms installation:

- .PRC file: C:\Satellite Forms 8\Extensions\<*creator name*>\Palm\
- .SFX file: C:\Satellite Forms 8\Extensions\<*creator name*>\PPC_ARM\
- .INF and .BMP files: C:\Satellite Forms 8\Extensions\<*creator name*>\

Under the Extensions directory, create a directory with the name of the creator of the extension (the company or developer name). In this directory, place the .INF and .BMP files – for SFX Custom controls – associated with your extension. By convention, preface the names of these files with `SFE_` for Satellite Forms Extension.

When you start Satellite Forms MobileApp Designer, it searches all the subdirectories of the Extensions directory for extensions to load into MobileApp Designer. If your extension is not in any of these subdirectories, you cannot access it from with MobileApp Designer.

Step 5: Load the extension into a Satellite Forms application

Follow the instructions under MobileApp Designer View menu on page 85 for adding a plug-in to a Satellite Forms application.

Step 6: Download the extension to the handheld device

Follow the instructions under MobileApp Designer Handheld menu on page 88 for downloading an extension to the handheld device.

Step 7: Test the extension by calling it from a Satellite Forms script

Follow the instructions under Creating a Satellite Forms script on page 309 for calling methods of a plug-in a Satellite Forms script.

Creating an SFX Custom control

While Satellite Forms provides a variety of standard user interface controls, some applications may require additional capabilities. Using the Satellite Forms API, you can create custom SFX Custom controls to meet virtually any user interface requirement. The following example illustrates how to create the Slider SFX Custom control supplied with Satellite Forms. Before working with this example, you should be familiar with the following concepts:

- Adding SFX Custom controls to a project and using them in Satellite Forms applications.
- Creating an SFX plug-in for Satellite Forms. For information, see the previous section, Creating an SFX plug-In on page 468.

Creating an SFX Custom control is similar to creating an SFX plug-in, with the addition of a few additional steps. Begin by reviewing the sample code from either the Generic or Square Root extensions. Beyond the code required to implement an SFX plug-in, SFX Custom controls have several additional requirements:

- Multiple instances of an SFX control can exist.
- SFX controls must handle system notification events in the `OnSfxNotify` event handler.
- SFX controls must read configuration information the user passes in.
- The .INF files for SFX controls contain additional information.

Review the Slider example source file, Main.c, for a complete example of how these requirements need to be implemented. If you installed Satellite Forms in the default installation directory, the source code for the Slider SFX Custom control extension is located in:
C:\Satellite Forms 8\Extensions\Slider\Src\

API function reference by category

The following sections document the Satellite Forms API by function category. For details on each function, including its usage, parameters, and return value, see the Alphabetical API Reference on page 481.

⚠ Caution API functions in *italics* are obsolete and kept for backward compatibility with only. These API functions may be removed from future versions of Satellite Forms without notice. Do not use these functions for new applications and replace them with the current functions in existing applications.

Memory allocation functions

The following table lists and describes memory allocation functions:

Table 12.3 Memory allocation functions

Function	**Description**
SF_AllocDbItem	Allocates a block of memory from the database heap and initializes it with the data passed in.
SF_db_free	Frees a memory block allocated with `SF_db_malloc`.
SF_db_malloc	Allocates a memory block from the database heap.
SF_db_realloc	Changes the size of a memory block allocated with `SF_db_malloc`.
SF_FreeCachedRecordData	Discards any memory associated with a cached record.
SF_xfree	Frees a memory block allocated with `SF_xmalloc`.
SF_xmalloc	Allocates a memory block from the dynamic heap.
SF_xrealloc	Changes the size of a memory block allocated with `SF_xmalloc`.

Table Operation functions

The following table lists and describes table operations functions:

Table 12.4 Table operation functions

Function	**Description**
SF_AppDesIndexToTable	Retrieves a pointer to the `TABLE_REC` from MobileApp Designer’s index of a Table object.s
SF_CommitCachedRecord	Saves data in the specified cached record to the record’s database.
SF_CommitItemToRow (Obsolete. Do not use.)	Stores data in a table. **Obsolete. Do not use.**
SF_CompareFields	Compares two Satellite Forms table fields and indicates whether they are equal or which is greater.
SF_CreateNewRecord	Creates a record in a table.

Table 12.4 Table operation functions *(Continued)*

SF_CurrentRowInvalid	Determines if the current record is invalid.
SF_DeleteRecord	Deletes a record from a table.
SF_DoTableLookup	Looks up the specified data in a table's key column and returns data in the corresponding return column.
SF_FindFirstRow	Returns the row number of the first record in a table.
SF_FindLastRow	Returns the row number of the last record in a table.
SF_FindNextRow	Returns the row number of the next record in a table.
SF_FindPrevRow	Returns the row number of the previous record in a table.
SF_GetActiveRecord	Retrieves the active record – the record the current form is displaying.
SF_GetCachedField	Retrieves a field's contents from a cached record.
SF_GetFieldCopy	Retrieves a copy of the contents of the specified field.
SF_GetFirstField (Obsolete. Do not use)	Internal function. **Obsolete. Do not use.**
SF_GetNumRows	Returns the number of records in a table.
SF_GetRowItemCopy (Obsolete. Do not use.)	Creates a copy of data. **Obsolete. Do not use.**
SF_GetTableColNumDecimals	Returns the number of decimal places in a numeric column.
SF_GetTableColType	Returns a column's type.
SF_GetTableColWidth	Returns the width of a column.
SF_LockRowItem (Obsolete. Do not use)	Internal function. **Obsolete. Do not use.**
SF_ResizeLockedRecord (Obsolete. Do not use.)	Internal function. **Obsolete. Do not use.**
SF_RowMeetsCriteria	Determines if a record meets the criteria of all active filters.
SF_SearchTable	Finds data in a table.
SF_SetCachedField	Replaces the contents of a field in a cached record.
SF_UnlockRowItem (Obsolete. Do not use)	Internal function. **Obsolete. Do not use.**

Form operation functions

The following table lists and describes form operations functions:

Table 12.5 Form operation functions

Function	Description
SF_AppDesIndexToForm	Retrieves a pointer to a Satellite Forms Form object structure from MobileApp Designer's index of a Form object.
SF_CommitFormToCurrentRow	Saves the data from all controls on the current page to the form's linked table.
SF_FormCreateRow	Creates and displays a record on a form.
SF_FormDeleteCurrRow	Deletes the current record from the form's linked table.
SF_FormDrawAll	Redraws the form.
SF_FormSlotAlloc	Allocates a slot for a new control in the Palm OS form structure.
SF_FormTableSizeChangedNotify	Notifies all controls on the current form that the size of the linked table has changed, enabling the controls to take appropriate action.
SF_FormValidate	Determines if all required fields on the current page contain data and all boundary conditions are satisfied.
SF_GetCurrentForm	Returns the current form.
SF_GetFirstForm	Returns the first form of the application.
SF_GetFormCurrentRow	Returns the row number of the current record displayed on a form.
SF_GetFormCurrPage	Returns the page number of the current page of the current form.
SF_GetFormFirstControl	Returns a pointer to the first control on a form.
SF_GetFormFlags	Returns the attribute flags for a form.
SF_GetFormNextForm	Returns a pointer to the next form in an application relative to the specified form.
SF_GetFormNumPages	Returns the number of pages in a form.
SF_GetFormOsFormPtr	Returns the Palm OS form that corresponds to a Satellite Forms form.
SF_GetFormReturnIndex	Returns the index of the form that called a specified form.
SF_GetFormTableIndex	Returns the index of a form's linked table.
SF_LoadFormWithCurrentRow	Loads the controls on the current page with data from the current record of the form's linked table.

Control operation functions

The following table lists and describes control operations functions:

Table 12.6 Control operation functions

Function	**Description**
SF_AppDesIndexToControlRec	Retrieves a pointer to Satellite Forms control from MobileApp Designer's index of a Control object.
SF_ClearInkRecord	Erases the contents of an ink control.
SF_ExecAutoStamp	Executes the intrinsic stamp action of an auto stamp control.
SF_GetControlBottom	Returns the bottom coordinate of a control.
SF_GetControlDataCopy	Copies the data in a control.
SF_GetControlFlags	Returns the attribute flags of a control.
SF_GetControlLeft	Returns the left coordinate of a control.
SF_GetControlNextControl	Returns the next control in a form relative to the specified control.
SF_GetControlOsIndex	Returns the Palm OS index of the specified Satellite Forms control.
SF_GetControlPageNum	Returns the page number where a control is located.
SF_GetControlRight	Returns the right coordinate of a control.
SF_GetControlTop	Returns the top coordinate of a control.
SF_GetControlType	Returns the type of a control.
SF_LoadCtrlObjFromCachedRecord	Loads a Check Box or Radio Button control with data from a cached record.
SF_LoadDropListFromCachedRecord	Loads a Drop List control with data from a cached record.
SF_LoadFieldFromCachedRecord	Loads an Edit control with data from a cached record.
SF_LoadInkFieldFromCachedRecord	Loads an Ink control with data from a cached record.
SF_LoadFormWithCurrentRow	Loads an Ink control with data from a record. **Obsolete. Do not use.**
SF_LockRecordAndCache	Locks the specified record and caches it in dynamic memory.
SF_QueryField	Provides a fast way to obtain a read-only pointer to a table.
SF_RenderInk	Redraws the contents of an Ink control.
SF_SaveCtrlObjToCachedRecord	Saves the contents of a Check Box or Radio Button control to a cached record.
SF_SaveDropListToCachedRecord	Saves the contents of a Drop List control to a cached record.
SF_SaveFieldToCachedRecord	Saves the contents of an Edit control to a cached record.

Table 12.6 Control operation functions *(Continued)*

SF_SaveInkFieldToCachedRecord	Saves the contents of an Ink control to a cached record.
SF_SetCtrlObjText	Sets the caption of a Check Box or Radio Button control.
SF_SetDropListText	Sets the caption of a Drop List control.
SF_SetFieldText	Sets the caption of an Edit control.
SF_SetLookupText	Sets the caption of a Lookup control.
SF_UnqueryField	Unlocks and releases a field that was accessed with `SF_QueryField`.
SF_ValidateField	Validates the contents of an Edit control to ensure the correct type.

Control action functions

The following table lists and describes control action functions:

Table 12.7 Control action functions

Function	Description
SF_AdvanceByPage	Moves among a form's pages and records.
SF_GetControlAction	Internal function. Not normally used.
SF_GotoNewForm	Jumps to a new form.
SF_GotoRow	Makes a record the current record.
SF_PerformControlAction	Executes the action associated with a control.

UI object conversion functions

The following table lists and describes UI object conversion functions:

Table 12.8 Object conversion functions

Function	Description
SF_ControlRecToOsObj	Converts a pointer to a Satellite Forms control to a pointer to the corresponding Palm OS control.
SF_GetFocusObjectPtr	Returns the Palm OS field object that has focus.
SF_GetUIObjectParent	Returns a pointer to the Satellite Forms control associated with a specified Palm OS control.
SF_IdToControlRec	Converts a Palm OS control ID to a pointer to a Satellite Forms control.
SF_IdToObjectPtr	Converts a Palm OS control ID to a pointer to a Palm OS control.
SF_OsIndexToControlRec	Converts an index in the Palm OS form's control array to a pointer to the corresponding Satellite Forms control.

Table 12.8 Object conversion functions *(Continued)*

SF_PointInControl	Determines if a point, identified by an (x,y) coordinate, lies within a control's rectangle.
SF_PointToControlRec	Converts a point, identified by an (x,y) coordinate, into a pointer to a control that is located at that position.

Format translation functions

The following table lists and describes format translation functions:

Table 12.9 Format translation functions

Function	Description
SF_ConvertDisplayToInternalFormat	Converts data in a format suitable for display to the user into the Satellite Forms internal table data format.
SF_ConvertInternalToDisplayFormat	Converts the Satellite Forms internal table data format into a format suitable for display to the user.
SF_FormatNumber	Formats a number.
SF_InternalToPilotDate	Converts the Satellite Forms internal representation of a date to the Palm OS representation.
SF_InternalToPilotTime	Converts the Satellite Forms internal representation of a time to the Palm OS representation.
SF_PilotDateToInternal	Converts the Palm OS representation of a date to the Satellite Forms internal representation.
SF_PilotTimeToInternal	Converts the Palm OS representation of a time to the Satellite Forms internal representation.

SFX extension initialization functions

The following table lists and describes SFX extension initialization functions:

Table 12.10 Extension initialization functions

Function	Description
SF_GetConfigVar	Returns the value of the specified key from the configuration section of the current instance of an SFX Custom control.
SF_GetExtensionControl	Returns a pointer to the Satellite Forms control associated with an SFX Custom control.
SF_GetGlobalPtr	Returns a pointer to the global data area used by an SFX Custom control.
SF_GetInstanceDataPtr	Returns a pointer to the instance data area used by an SFX Custom control.
SF_InstallHandler	Installs a handler for an event.
SF_SetGlobalPtr	Saves a pointer to the global data area an SFX Custom control uses.

Table 12.10 Extension initialization functions *(Continued)*

SF_SetInstanceDataPtr	Saves a pointer to the instance data area used by an SFX Custom control.

Floating-point operation functions

The following table lists and describes floating-point operations functions:

Table 12.11 Floating-point operation functions

SF_FloatAdd	**Performs floating-point addition.**
SF_FloatEq	Evaluates whether two floating-point numbers are equal.
SF_FloatDiv	Performs floating-point division.
SF_FloatGe	Evaluates whether one floating-point number is greater than or equal to another.
SF_FloatGt	Evaluates whether one floating-point number is greater than another.
SF_FloatLe	Evaluates whether one floating-point number is less than or equal to another.
SF_FloatLt	Evaluates whether one floating-point number is less than another.
SF_FloatNe	Evaluates whether two floating-point numbers are not equal.
SF_FloatMul	Performs floating-point multiplication.
SF_FloatSub	Performs floating-point subtraction.

Scripting functions

The following table lists and describes scripting functions:

Table 12.12 Scripting functions

Function	**Description**
SF_Beep	Issues a beep from the handheld device speaker.
SF_ColumnSum	Sums data values in a specified column.
SF_GetSysTime	Returns the number of seconds since January 1, 1904.
SF_InternalDateToSysDate	Converts the Satellite Forms representation of a date to the scripting language representation.
SF_InternalTimeToSysTime	Converts the Satellite Forms representation of a time to the scripting language representation.
SF_ScriptExecExt	Executes an extension's method or accesses a property.
SF_ScriptFloatToString	Converts a floating-point value into a string.
SF_ScriptFreeParamMem	Internal function. Not normally used.

Table 12.12 Scripting functions *(Continued)*

SF_ScriptGetTosPtr	Returns a pointer to the item on the top of the stack.
SF_ScriptPopFloat	Removes an item from the top of the stack and returns it as a floating-point number.
SF_ScriptPopInt64	Removes an item from the top of the stack and returns it as a 64-bit integer.
SF_ScriptPopInteger	Removes an item from the top of the stack and returns it as an integer.
SF_ScriptPopString	Removes an item from the top of the stack and returns it as a string.
SF_ScriptPushFloat	Pushes a floating-point value onto the stack.
SF_ScriptPushFloatFromStr	Pushes a floating-point value onto the stack from a string value.
SF_ScriptPushInt64	Pushes a 64-bit integer onto the stack.
SF_ScriptPushInteger	Pushes an integer onto the stack.
SF_ScriptPushStaticStr	Pushes a string that was **not** allocated with `SF_xmalloc` onto the stack.
SF_ScriptPushVar	Pushes a variable onto the stack.
SF_ScriptPushVarString	Pushes a string allocated with `SF_xmalloc` onto the stack.
SF_TaskDelay	Puts the handheld device in energy-conserving doze mode for the specified duration.
SF_Tone	Issues a tone of a specified frequency, duration, and amplitude.
SF_ScriptVarAssign	Assigns the value on the top of the stack to a variable.
SF_SysDateToInternalDate	Converts the scripting language representation of a date to the Satellite Forms internal representation.
SF_SysTimeToInternalTime	Converts the scripting language representation of a time to the Satellite Forms internal representation.

Message and error functions

The following table lists and describes message and error functions:

Table 12.13 Message and error functions

Function	Description
SF_AssertFail	Displays a dialog box indicating an assertion failure at a specified file and line number.
SF_Beep	Issues an error beep from the handheld device speaker.
SF_ConfirmMsg	Displays a dialog box containing OK and Cancel buttons.
SF_InfoMsg	Displays a dialog box containing a message and an OK button.

Table 12.13 Message and error functions *(Continued)*

SF_InfoMsgInt	Displays a dialog box containing a message followed by an integer.
SF_ShowAbout	Displays the Satellite Forms About box.

Miscellaneous functions

The following table lists and describes miscellaneous functions:

Table 12.14 Miscellaneous functions

Function	Description
SF_atoi	Converts a string to an integer.
SF_CloneString	Creates a copy of a string.
SF_DeleteApplication	Deletes an application and all of its tables from the handheld device.
SF_DeleteDatabaseByName	Deletes a file from the handheld device.
SF_DoButtonBehavior	Causes a rectangle to behave like a Palm OS button.
SF_GetEngineVersion	Retrieves the full version number of the Satellite Forms engine.
SF_itoa	Converts a string to an integer.
SF_itoh	Converts an integer to a hexadecimal ASCII string.
SF_memcpy	???
SF_memset	Fills a range of dynamic memory range with the specified value.
SF_memsize	Returns the size of the specified pointer.
SF_strcat	Concatenates a string to another.
SF_strchr	Searches a string for the specified character.
SF_strcmp	Case-sensitive comparison of two strings.
SF_strcpy	Copies one string to another.
SF_stricmp	Case-insensitive comparison of two strings.
SF_strlen	Returns the length of the specified string.
SF_strstr	Searches for a sub-string within a string.

Alphabetical API Reference

The following sections provide an alphabetical listing of the Satellite Forms API functions including usage, parameters, return values of each function.

For information on which API functions to use for specific tasks, see the previous section, API function reference by category on page 472.

SF_AdvanceByPage

CBOOL SF_AdvanceByPage (WORD *MoveType*)

Moves among a form's pages and the records of its linked table.

Parameter	*MoveType*	The kind of move to make.
Return Value	TRUE if the move is successful; FALSE if it is not.	
Comments	The *MoveType* parameter takes the following values:	
	ACTIONTYPE_FIRST	Moves to the first page and the first record.
	ACTIONTYPE_LAST	Moves to the last page and the last record.
	ACTIONTYPE_NEXT	Moves to the next page. On the last page, moves to the next record.
	ACTIONTYPE_PREV	Moves to the previous page. On the first page, moves to the previous record.

Using this function with ACTIONTYPE_NEXT fails if the current page is the last page and the current record is the last record. Similarly, using this function with ACTIONTYPE_PREV fails if the current page is the first page and the current record is the first record.

SF_AllocDbItem

DB_ITEM * SF_AllocDbItem (void **pData,* WORD *DataLen*)

Allocates a block of memory from the database heap and initializes it with the data passed in.

Parameters	*pData*	Pointer to the data to be copied into the newly allocated block of memory.
	DataLen	Size of data *pData* points to.

Return Value A pointer to the newly allocated and initialized block of memory or NULL if the function fails

Comments A DB_ITEM is a Satellite Forms-specific database item that represents a single field value. A record in a Satellite Forms database is composed of several DB_ITEM structures placed sequentially within the record. Functions that manipulate the contents of record fields typically use DB_ITEM to represent the field data. Note: The exact format of DB_ITEM is not important. Treat DB_ITEM as an opaque data structure similar to Handles in the Win32 API. Satellite Forms API functions generate and consume these structures. Simply pass the DB_ITEM structures between API functions.

SF_AppDesIndexToControlRec

CONTROL_REC * SF_AppDesIndexToControlRec (Word *Index*)

Retrieves a pointer to a Satellite Forms control from MobileApp Designer's index of a Control object.

Parameter *Index* Zero-based index of the desired control.

Return Value Pointer to a Satellite Forms control from MobileApp Designer's index of the Control object.

Comments The prototype for this function is in the Include file SFExt.h.

This function converts a MobileApp Designer index to a pointer to a control. Indexes are typically passed to an extension through a configuration variable. A control index, when passed through a configuration variable, looks like this: *%Cnnn*. The *nnn* is the MobileApp Designer index. You can convert the index to an integer by passing a pointer to the *nnn* part to `SF_atoi`.

For more information, see the Slider SFX Custom control example in the following directory:
C:\Satellite Forms 8\Samples\Extensions\Slider\Src\

See Also SF_AppDesIndexToForm, SF_APPDESINDEXTOTABLE

SF_AppDesIndexToForm

FORM_HEADER * SF_AppDesIndexToForm (Word *Index*)

Retrieves a pointer to a Satellite Forms Form object from MobileApp Designer's index of a Form object.

Parameter *Index* Zero-based index of the desired form.

Return Value Pointer to a Satellite Forms Form object from MobileApp Designer's index of a Form object

Comments The prototype for this function is in the Include file SFExt.h.

This function converts an MobileApp Designer index to a pointer to a form. Indexes are typically passed to an extension through a configuration variable. A control index, when passed through a configuration variable, looks like this: *%Fnnn*. The *nnn* is the App Designer index. You can convert it to an integer by passing a pointer to the *nnn* part to `SF_atoi`.

For more information, see the Slider SFX Custom control example in the following directory:
C:\Satellite Forms 8\Samples\Extensions\Slider\Src\

See Also SF_APPDESINDEXTOCONTROLREC, SF_APPDESINDEXTOTABLE

SF_AppDesIndexToTable

TABLE_REC * SF_AppDesIndexToTable (Word *Index*)

Retrieves a pointer to TABLE_REC from MobileApp Designer's index of a Table object.

Parameter *Index* Zero-based index of the desired table.

Return Value Pointer to TABLE_REC from MobileApp Designer's index of a Table object.

Comments The prototype for this function is in the Include file SFEXT.H.

This function converts a MobileApp Designer index to a pointer to a table. Indexes are typically passed to an extension through a configuration variable. A control index is passed through a configuration variable that looks like this: *%Tnnn*. The *nnn* is the App Designer index. You can convert it to an integer by passing a pointer to the *nnn* part to `SF_atoi`.

For more information, see the Slider SFX Custom control example in the following directory:
C:\Satellite Forms 8\Samples\Extensions\Slider\Src\

The file Main.c has an example of using the `SF_AppDesIndexToTable` function. A field is passed to this control as a configuration variable.

See Also SF_APPDESINDEXTOCONTROLREC, SF_AppDesIndexToForm

SF_AssertFail

Void SF_AssertFail (char **pFile,* int *LineNum*)

Displays a dialog box indicating an assertion failure at a specified file and line number.

Parameters *pFile* Pointer to a string containing the name of a file that is displayed in the dialog box.

LineNum Line number displayed in the dialog box.

Return Value None

Comments With Metrowerks' CodeWarrior, you can use the pre-defined compiler symbols `__FILE__` for the filename and `__LINE__` for the line number.

SF_atoi

long SF_atoi (const char **pszStr*)

Converts a string to an integer.

Parameter *pszStr* Pointer to a string to convert.

Return Value The converted integer.

SF_Beep

void SF_Beep (Word *BeepType*)

Issues a beep from the handheld device speaker.

Parameter *BeepType* Type of beep.

Return Value None

Comments The *BeepType* parameter takes the following values:

1 = Information
2 = Warning
3 = Error
4 = Startup
5 = Alarm
6 = Confirmation
7 = Click

See Also SF_Tone

SF_ClearInkRecord

void SF_ClearInkRecord (CONTROL_REC **pControl*)

Erases the contents of an Ink control.

Parameter *pControl* Pointer to the desired Ink control.

Return Value None

SF_CloneString

char * SF_CloneString (char **pStr*)

Creates a copy of a string.

Parameter *pStr* Pointer to the string to be copied.

Return Value Pointer to a string containing the copy of the input string.

Comments This function allocates memory for the cloned string with `SF_xmalloc`. Free the allocated memory with `SF_xfree` when done with the string.

SF_ColumnSum

DWORD SF_ColumnSum (WORD *TableIndex,* WORD *ColumnIndex*)

This function sums data values in a table's column.

Parameters	*TableIndex*	Index of the desired table.
	ColumnIndex	Index of the desired column in the specified table.

Return Value Sum of the data in the specified column.

Comments Only columns that match the criteria of all active filters are summed.

SF_CommitCachedRecord

CBOOL SF_CommitCachedRecord (CACHED_RECORD **pCachedRec*)

Save the data in the specified cached record into the record's database.

Parameter *pCachedRec* Pointer to the desired cached record.

Return Value TRUE if successful; FALSE if the function fails.

Comments Typically, the record being displayed in and operated on by the current form is cached in memory. Internal functions use `SF_CommitCachedRecord` to save the contents of the cached record back to its original database.

SF_CommitFormToCurrentRow

CBOOL SF_CommitFormToCurrentRow ()

Saves the data from all controls on the current page to the form's underlying table.

Parameters None

Return Value TRUE if the data passes validation and is saved to the underlying table; FALSE if the function fails.

SF_CommitItemToRow(Obsolete. Do not use.)

void SF_CommitItemToRow (TABLE_REC **pRec,* WORD *RowNum,* WORD *ColIndex,* BYTE **pItem*)

Stores data in a table.

Parameters	*pRec*	Pointer to the desired table.
	RowNum	Zero-based row number of the record in which data is to be stored.
	ColIndex	Index of the column in which the data is to be stored.
	pItem	Pointer to the data to be stored.

Return Value None

Comments The data item must be a null-terminated string. It does not have to be as large as the column width. If the data item is numeric, it must have the correct number of decimal places. For example, if there are two decimal places, you must express 1 as 1.00.

See Also SF_GetRowItemCopy (Obsolete. Do not use.)

SF_CompareFields

short SF_CompareFields (DB_ITEM **pField1,* DB_ITEM **pField2,* TABLE_REC **pRec,* WORD *ColIndex*)

Compares two Satellite Forms database fields and indicates whether they are equal or which is greater.

Parameters	*pField1*	Pointer to the first field.
	pField2	Pointer to the second field.
	pRec	Pointer to the table containing the first field.
	ColIndex	Index of the first field's column.

Return Value This function returns zero if the two fields are equal, 1 if the first field is greater than the second, and –1 otherwise.

Comments The two fields must be of the same data type.

SF_ConfirmMsg

CBOOL SF_ConfirmMsg (char **p*)

Displays a dialog box containing a message and OK and Cancel buttons.

Parameter	*p*	Pointer to a string containing the message.

Return Value TRUE if the user taps OK; FALSE if the user taps Cancel.

See Also SF_InfoMsg, SF_InfoMsgInt

SF_ControlRecToOsObj

void * SF_ControlRecToOsObj (CONTROL_REC **pControl,* FormObjectKind *ObjType*)

Converts a pointer to a Satellite Forms control to a pointer to the corresponding Palm OS control.

Parameters	*pControl*	Pointer to the desired a Satellite Forms control.
	ObjType	Palm OS object type.

Return Value Pointer to the corresponding Palm OS control.

Comments The *ObjType* parameter is used to validate the object type of the control. You can pass in –1 if you do not want to validate the object type.

SF_ConvertDisplayToInternalFormat

BYTE * SF_ConvertDisplayToInternalFormat (const char **pszItem*, const TABLE_REC **pRec*, WORD *wColIndex*, WORD **pwFldDataSize*)

Converts data in a format suitable for display to the user to the Satellite Forms internal table data format.

Parameters *pszItem* String to be converted into internal format.

pRec Pointer to the table to which the data is saved.

wColIndex Column index that represents the data format.

pwFldDataSize Buffer that holds the size of the data returned by this function.

Return Value Null or the converted data allocated in the heap. The caller must free the allocated memory using `SF_xfree`.

See Also SF_ConvertInternalToDisplayFormat

SF_ConvertInternalToDisplayFormat

BYTE * SF_ConvertInternalToDisplayFormat (const DB_ITEM *_pItem_, WORD *_pwSize_, const TABLE_REC *_pRec_, WORD _wColIndex_)

Converts the Satellite Forms internal table data format into a format suitable for display to the user.

Parameters *pItem* DB_ITEM returned by API function such as `GetCachedField`.

pwSize Buffer to hold the size of the returned string.

pRec Pointer to the table from which the data was retrieved.

wColIndex Column index that represents the data format.

Return Value Null or a string allocated in the heap. The caller must free the string using `SF_xfree`.

See Also *SF_GetRowItemCopy (Obsolete. Do not use.)*, SF_ConvertDisplayToInternalFormat

SF_CreateNewRecord

WORD SF_CreateNewRecord (TABLE_REC *_*pRec,_* CBOOL *fPromptUser*)

Creates a record in the specified table.

Parameters *pRec* Pointer to the desired table.

fPromptUser Boolean value specifying whether to prompt the user to confirm the record creation: TRUE prompts; FALSE does not prompt.

Return Value Zero-based row number of the new record if successful; 0xFFFF if the function failed to create the new record.

Comments If *fPromptUser* is TRUE, this function prompts the user to confirm creation of the new record. If the user does not allow the record to be created, the function fails.

If *fPromptUser* is FALSE, a record is created without prompting the user.

New records are added to the end of the table. It is not possible to insert a new record into any other position in a table.

See Also SF_DeleteRecord, SF_FormCreateRow, SF_FormDeleteCurrRow

SF_CurrentRowInvalid

CBOOL SF_CurrentRowInvalid ()

Determines if the current record – the record being displayed by the current form – is invalid.

Parameters None

Return Value TRUE if there is no table linked to the current form, if the table linked to the current form has no records, or if active filters exclude all records; FALSE if the row is valid.

Comments In general, you should use this function before manipulating data in the current record.

SF_db_free

void SF_db_free (void **pMem*)

Frees a block of memory allocated with `SF_db_malloc`.

Parameter	*pMem*	Pointer to block of memory allocated with `SF_db_malloc`.
Return Value	None	

Comments `SF_db_malloc`, `SF_db_realloc`, and `SF_db_free` manipulate memory on the database heap. Use these functions whenever you need to allocate large amounts of memory.

Memory allocated using these functions is read-only when accessed through a pointer. To write to this memory, you must use the Palm OS database memory-write functions, for example, `DmSet`, `DmWrite`.

See Also SF_db_malloc, SF_db_realloc, SF_xfree, SF_xmalloc, SF_xrealloc

SF_db_malloc

void * SF_db_malloc (WORD *Size*)

Allocates a block of memory from the database heap.

Parameter	*Size*	Requested size of the memory block.

Return Value Pointer to the allocated memory block if successful; NULL if the function failed to allocate the requested memory.

Comments `SF_db_malloc`, `SF_db_realloc`, and `SF_db_free` manipulate memory from the database heap. Use these functions whenever you need to allocate large amounts of memory.

Memory allocated using these functions is read-only when accessed through a pointer. To write to this memory, you must use the Palm OS database memory-write functions, for example, `DmSet`, `DmWrite`.

See Also SF_db_free, SF_db_realloc, SF_xfree, SF_xmalloc, SF_xrealloc

SF_db_realloc

void * SF_db_realloc (void **pMem,* WORD *NewSize*)

Changes the size of a memory block allocated with SF_DB_MALLOC.

Parameter	*pMem*	Pointer to a block of memory allocated with `SF_db_malloc`
	NewSize	New requested size of the specified memory block.

Return Value Pointer to the reallocated memory block if successful; NULL if the function fails to reallocate the specified block of memory.

Comments The block of memory may move as a result of the reallocation.

`F_db_malloc`, `SF_db_realloc`, and `SF_db_free` manipulate memory from the database heap. Use these functions whenever you need to allocate large amounts of memory.

Memory allocated using these functions is read-only when accessed through a pointer. To write to this memory, you must use the Palm OS database memory-write functions, for example, `DmSet`, `DmWrite`.

See Also SF_db_malloc, SF_db_free, SF_xfree, SF_xmalloc, SF_xrealloc

SF_DeleteApplication

void SF_DeleteApplication (char **pAppName*)

Deletes an application and all of its tables from the handheld device.

Parameter *pAppName* The name of the file containing the application to be deleted.

Return Value None

Comments Satellite Forms application filenames are in the form SF_*, for example, SF_MyApp.

See Also SF_DeleteDatabaseByNamec

SF_DeleteDatabaseByName

CBOOL SF_DeleteDatabaseByName (char **pName*)

Deletes a database file from the handheld device.

Parameter *pName* The name of the database file to be deleted.

Return Value TRUE if the database file is successfully deleted; FALSE if the function fails to delete the database file.

Comments **Caution:** This function can delete any file on the handheld device, not just Satellite Forms database files.

See Also SF_DeleteApplication

SF_DeleteRecord

CBOOL SF_DeleteRecord (TABLE_REC **pRec,* WORD *RowNum,* CBOOL *fPromptUser*)

Deletes a record from the specified table.

Parameters *pRec* Pointer to the desired table.

RowNum Zero-based row number of record to be deleted.

fPromptUser Boolean value specifying whether to prompt the user to confirm the record deletion: TRUE prompts; FALSE does not prompt.

Return Value TRUE if the record is successfully deleted; FALSE if the function fails to delete the record.

Comments If *fPromptUser* is TRUE, this function prompts the user to confirm deletion of the record. If the user does not allow the record to be deleted, the function fails.

If *fPromptUser* is FALSE, the record is deleted without prompting the user.

If you delete the current row, you must reset the current row using, for example, `SF_GotoRow` with the *fSaveCurrent* parameter set to FALSE.

See Also SF_CreateNewRecord, SF_FormDeleteCurrRow, SF_GotoRow

SF_DoButtonBehavior

CBOOL SF_DoButtonBehavior (RectangleType **pRect*)

Causes a rectangle to behave like a Palm OS button.

Parameter *pRect* Pointer to the desired rectangle.

Return Value TRUE if the user lifts the stylus inside the rectangle; FALSE otherwise.

Comments When you receive a `PenDownEvent` within the bounds of the specified rectangle, call this function. It tracks the stylus until the user lifts the stylus. As the stylus enters the rectangle, the rectangle inverts. As the stylus leaves the rectangle, the rectangle will revert to normal. If the user lifts the stylus within the bounds of the rectangle, this function returns TRUE.

SF_DoTableLookup

DB_ITEM * SF_DoTableLookup (DB_ITEM **pKeyItem,* WORD *TableIndex,* WORD *KeyColIndex,* WORD *RetColIndex*)

Looks up the specified data in a table's key column and returns data in the corresponding return column.

Parameters *pKeyItem* Pointer to data to be looked up – a zero-terminated string.

TableIndex Index of the desired table.

KeyColIndex Index of the column in which data will be looked up.

RetColIndex Index of the column from which data is to be returned.

Return Value Pointer to a copy of the data stored in the column *RetColIndex* for the first row where the data referenced by *pKeyItem* is found in the column *KeyColIndex* if successful; NULL otherwise.

Comments This function allocates the copy of the data using `SF_xmalloc`. The caller must free this memory with `SF_xfree` when finished using it.

See Also SF_SearchTable

SF_ExecAutoStamp

void SF_ExecAutoStamp (CONTROL_REC **pControl*)

Executes the intrinsic stamp action of an Auto Stamp control.

Parameter *pControl* Pointer to the desired Auto Stamp control

Return Value None

Comments Use this function to log the current date and time in records.

SF_FindFirstRow

WORD SF_FindFirstRow (TABLE_REC **pRec*)

Returns the row number of the first record of a table.

Parameter *pRec* Pointer to the desired table.

Return Value Zero-based row number of the first record in the table if successful; 0xFFFF if the function fails.

Comments The first record may not be record 0 if there are filters applied to the table.

This function fails if there are no records in the table or if there are no records that satisfy the criteria of all active filters.

See Also SF_FindLastRow, SF_FindNextRow, SF_FindPrevRow

SF_FindLastRow

WORD SF_FindLastRow (TABLE_REC **pRec*)

Returns the row number of the last record of a table.

Parameter *pRec* Pointer to the desired table.

Return Value Zero-based row number of the last record in the table if successful; 0xFFFF if the function fails.

Comments This function fail if there are no records in the table or if there are no records that satisfy the criteria of all active filters.

See Also SF_FindFirstRow, SF_FindNextRow, SF_FindPrevRow

SF_FindNextRow

WORD SF_FindNextRow (TABLE_REC **pRec,* WORD *RowNum*)

Returns the row number of the next record of a table.

Parameters *pRec* Pointer to the desired table.

RowNum Zero-based row number of the current record.

Return Value Zero-based row number of the next record in the table, as referenced from *RowNum*, if successful; 0xFFFF if the function fails.

Comments This function fails if there is no next record – record *RowNum* is the last record in the table or is the last record in the table that satisfies the criteria of all active filters.

See Also SF_FindFirstRow, SF_FindLastRow, SF_FindPrevRow

SF_FindPrevRow

WORD SF_FindPrevRow (TABLE_REC **pRec,* WORD *RowNum*)

Returns the row number of the previous record of a table.

Parameters *pRec* Pointer to the desired table.

RowNum Zero-based row number of the current record.

Return Value Zero-based row number of the previous record in the table, as referenced from *RowNum*, if successful; 0xFFFF if the function fails.

Comments This function fails if there is no next record – record *RowNum* is the first record in the table or is the first record in the table that satisfies the criteria of all active filters.

See Also SF_FindFirstRow, SF_FindLastRow, SF_FindNextRow

SF_FloatAdd

FlpDouble SF_FloatAdd (FlpDouble *n1*, FlpDouble *n2*)

Adds two floating-point numbers.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value Floating-point result of the addition.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatDiv, SF_FloatMul, SF_FloatSub

SF_FloatDiv

FlpDouble SF_FloatDiv (FlpDouble *n1*, FlpDouble *n2*)

Divides one floating-point number by another.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value Floating-point result of division of *n1* / *n2*

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatAdd, SF_FloatMul, SF_FloatSub

SF_FloatEq

CBOOL SF_FloatEq (FlpDouble *n1*, FlpDouble *n2*)

Evaluates whether two floating-point numbers are equal.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value TRUE if *n1* and *n2* are equal; FALSE if they are not equal.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatGe, SF_FloatGt, SF_FloatLe, SF_FloatLt, SF_FloatNe

SF_FloatGe

CBOOL SF_FloatGe (FlpDouble *n1*, FlpDouble *n2*)

Evaluates whether one floating-point number is greater than or equal to another.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value TRUE if *n1* is greater than or equal to *n2*; FALSE if not.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatEq, SF_FloatLe, SF_FloatLt, SF_FloatNe

SF_FloatGt

CBOOL SF_FloatGt (FlpDouble *n1*, FlpDouble *n2*)

Evaluates whether one floating-point number is greater than another.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value TRUE if *n1* is greater than *n2*; FALSE if not.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatEq, SF_FloatGe, SF_FloatLe, SF_FloatLt, SF_FloatNe

SF_FloatLe

CBOOL SF_FloatLe (FlpDouble *n1*, FlpDouble *n2*)

Evaluates whether one floating-point number is less than or equal to another.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value TRUE if *n1* is less than or equal to *n2*; FALSE if not.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatEq, SF_FloatGe, SF_FloatGt, SF_FloatLt, SF_FloatNe

SF_FloatLt

CBOOL SF_FloatLt (FlpDouble *n1*, FlpDouble *n2*)

Evaluates whether one floating-point number is less than another.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value TRUE if *n1* is less than *n2*; FALSE if not.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatEq, SF_FloatGe, SF_FloatGt, SF_FloatLe, SF_FloatNe

SF_FloatMul

FlpDouble SF_FloatMul (*FlpDouble n1*, *FlpDouble n2*)

Multiplies two floating-point numbers.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value Floating-point result of the multiplication.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatAdd, SF_FloatDiv, SF_FloatSub

SF_FloatNe

CBOOL SF_FloatNe (FlpDouble *n1*, FlpDouble *n2*)

Evaluates whether two floating-point numbers are not equal.

Parameters *n1* Floating-point number

n2 Floating-point number

Return Value TRUE if *n1* and *n2* are not equal; FALSE if they are equal.

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatEq, SF_FloatGe, SF_FloatGt, SF_FloatLe, SF_FloatLt

SF_FloatSub

FlpDouble SF_FloatSub (FlpDouble *n1*, FlpDouble *n2*)

Subtracts one floating-point number from another.

Parameters *n1* Floating-point number.

n2 Floating-point number.

Return Value Floating-point result of the subtraction *n1* – *n2.*

Comments The floating-point functions are present so that extensions that have simple floating-point requirements can use the Satellite Forms engine OS-independent routines. If your extension requires additional floating-point functionality, you must use OS routines appropriate to the version of the OS under which your extension is running.

See Also SF_FloatAdd, SF_FloatDiv, SF_FloatMul

SF_FormatNumber

CBOOL SF_FormatNumber (char **pNum,* BYTE *MaxLen,* BYTE *NumDec*)

Formats a number. Removes leading zeros and adds trailing zeros in decimal places if necessary.

Parameters *pNum* In/Out. Pointer to a string containing the number to be formatted.

MaxLen In. Maximum size of formatted number.

NumDec In. Number of decimal places.

Return Value TRUE if the formatting is successful; FALSE if the function fails to format the number.

Comments This function performs an in-place conversion. *pNum* points to the newly formatted number. Some examples of this function’s behavior appear below:

pNum In	*MaxLen*	*NumDec*	*pNum* Out	Return Value

1	5	2	“1.00”	TRUE
1.372	4	2	“1.37”	TRUE
1.372	4	3	“1.37”	FALSE

Note that in the third example the number of decimal places was reduced from that specified in *NumDec* in order to meet the criteria specified in *MaxLen*. *MaxLen* always takes precedence over *NumDec*.

This function returns FALSE in this case because it was not able to fit three decimal places into a total of four spaces. The buffer referenced by *pNum* receives the converted number. It must be at least *MaxLen* + 1 bytes long to receive the terminating zero of the string.

This function fails if the number contains invalid characters or does not fit in the specified buffer.

SF_FormCreateRow

void SF_FormCreateRow (CBOOL *fPromptUser*)

Creates a record in a form and displays this record.

Parameter *fPromptUser* Boolean value specifying whether to prompt the user to confirm the record creation: TRUE prompts; FALSE does not prompt.

Return Value None

Comments This function does everything required to create a new record and change the form’s current record to the new record. Before creating the new record, this function validates and saves the form’s current record.

See Also SF_CreateNewRecord, SF_DeleteRecord, SF_FormDeleteCurrRow

SF_FormDeleteCurrRow

void SF_FormDeleteCurrRow (CBOOL *fPromptUser*)

Deletes the current record from the form’s linked table.

Parameter *fPromptUser* Boolean value specifying whether to prompt the user to confirm the record deletion: TRUE prompts; FALSE does not prompt.

Return Value None

Comments This function issues a beep if you try to delete a non-existent record.

`SF_FormDeleteCurrRow` differs from `SF_DeleteRecord` in that it automatically changes the form’s current record to a valid record after completing the deletion from the table. If you use `SF_DeleteRecord`, you must manually change the form’s current record.

See Also SF_CreateNewRecord, SF_DeleteRecord, SF_FormCreateRow, SF_FormTableSizeChangedNotify, SF_LoadFormWithCurrentRow

SF_FormDrawAll

void SF_FormDrawAll (CBOOL *fMinimal*)

Redraws the form.

Parameter *fMinimal* Boolean value specifying whether all or only parts of the form should be redrawn.

Return Value None

Comments Always set *fMinimal* to FALSE to ensure that the entire form is redrawn.

SF_FormSlotAlloc

WORD SF_FormSlotAlloc (FormPtr *frm*)

Allocates a slot for a new control in the OS form structure.

Parameter *frm* Pointer to the OS form.

Return Value The index of the slot that was created if successful; 0xFFFF if the function fails.

SF_FormTableSizeChangedNotify

void SF_FormTableSizeChangedNotify (CBOOL *fRedrawControl*)

Notifies all controls on the current form that the size of the linked table has changed, enabling the controls to take appropriate action.

Parameter *fRedrawControl* Boolean value specifying whether to redraw the controls based on the table size change.

Return Value None

Comments A table's size changes when records are added or deleted. It is not usually necessary to use this function because `SF_CreateNewRecord`, `SF_DeleteRecord`, `SF_FormCreateRow`, and `SF_FormDeleteCurrRow` call this function.

SF_FormValidate

CBOOL SF_FormValidate ()

Determines whether all required fields on the current page are filled and all boundary conditions are satisfied.

Parameter None

Return Value TRUE if all required fields are filled and all boundary conditions satisfied; FALSE if any required field is not filled or any boundary condition is not satisfied

Comments If the offending field is an input control, the cursor (caret) is placed in the field.

See Also SF_ValidateField

SF_FreeCachedRecordData

void SF_FreeCachedRecordData *(*CACHED_RECORD **pCachedRec)*

Discards memory associated with a cached record.

Parameter *pCachedRec* Pointer to the desired cached record.

Return Value None

Comments If you explicitly cache a record using `SF_LockRecordAndCache`, you must remember to free the record's cached data using this function. Failure to do so causes a memory leak.

See Also SF_LockRecordAndCache

SF_GetActiveRecord

CACHED_RECORD * SF_GetActiveRecord ()

Retrieves the active record from a cached record.

Parameters None

Return Value Returns a pointer to the active record; NULL if there is no active record.

Comments The active record is the record currently displayed on the current form. If the form is not linked to a table, `SF_GetActiveRecord` returns NULL. The Satellite Forms engine caches the current record, keeping a copy in the dynamic heap, for performance reasons.

It is important to understand this concept because you must take into consideration whether a record is cached or not when editing a table record directly.

For example, if the record you would like to edit with a script is the active record, you should edit the *cached* record instead of the record stored in the database. If you edit the table record directly, your changes are overwritten when you move the form to another record.

SF_GetCachedField

DB_ITEM * SF_GetCachedField (CACHED_RECORD **pCachedRec,* WORD *ColIndex*)

Retrieves a field's contents from a cached record.

Parameters *pCachedRec* Pointer to the desired cached record.

ColIndex Index of the column containing the data to retrieve.

Return Value A pointer to a `Db_Item` structure containing the field's data; NULL if the function fails.

Comments The pointer may be either to a block on the dynamic heap or a block on the database heap.

The field is part of the record and therefore should not be freed individually when you are finished using it. Instead, if you cached the record yourself using `SF_LockRecordAndCache`, free all data in one operation using `SF_FreeCachedRecordData`.

Use the data in this field as read-only. If you want to modify the contents of the field, use `SF_SetCachedField` to replace the existing field in the cached record.

The format of the data returned depends on the field type. For more information, see `SF_SetCachedField`.

See Also SF_AllocDbItem, SF_SetCachedField, SF_GetFieldCopy, SF_FreeCachedRecordData, SF_LockRecordAndCache

SF_GetConfigVar

char * SF_GetConfigVar (char **pName*)

Returns the value of the specified key from the configuration section of the current instance of an SFX Custom control.

Parameter *pName* Pointer to a zero-terminated key name in the configuration section of an SFX Custom control.

Return Value The value of the key if found; NULL if not found or if the control is not an SFX Custom control

Comments The string that *pName* points to **must be upper case**.

SF_GetControlAction

CONTROL_ACTION * SF_GetControlAction (CONTROL_REC **pControl*)

Returns a pointer to the data structure associated with a control's actions and filters. This is an internal function and should not normally be used.

Parameter *pControl* Pointer to the desired control.

Return Value Pointer to the data structure associated with the control.

Comments This function is only used with `SF_PerformControlAction`.

See Also SF_PerformControlAction

SF_GetControlBottom

WORD SF_GetControlBottom (CONTROL_REC **pControl*)

Returns the bottom coordinate of a control.

Parameter *pControl* Pointer to the desired control.

Return Value The *y* coordinate of the bottom of the control's rectangle.

Comments The origin is the upper-left corner of the screen; all coordinates are expressed in pixels.

See Also SF_GetControlLeft, SF_PointInControl, SF_GetControlRight, SF_GetControlTop

SF_GetControlDataCopy

void * SF_GetControlDataCopy (CONTROL_REC **pControl*)

Creates a copy of the data in a control.

Parameter *pControl* Pointer to the desired control.

Return Value A pointer to a copy of the data if successful; NULL if the function fails.

Comments The data is a zero-terminated string. This function allocates memory using `SF_xmalloc`. Free this memory using `SF_xfree`.

SF_GetControlFlags

DWORD SF_GetControlFlags (CONTROL_REC **pControl*)

Returns the attribute flags of a control.

Parameter *pControl* Pointer to the desired control.

Return Value Control attribute flags.

Comments The following control flags are defined in SFDefs.h:

`CFLAG_COMMON_READONLY`: Read-Only

`CFLAG_COMMON_NOSAVE`: Do not Modify Table

`CFLAG_COMMON_ENABLED`: Visible

`CFLAG_COMMON_ALTSHAPE`: Alternate Shape

`CFLAG_COMMON_RIGHTANCHOR`: Right Anchor

`CFLAG_COMMON_REQUIRED`: Input Required

SF_GetControlLeft

WORD SF_GetControlLeft (CONTROL_REC **pControl*)

Returns the left coordinate of a control.

Parameter *pControl* Pointer to the desired control.

Return Value The *x* coordinate of the left side of the control's rectangle.

Comments The origin is the upper-left corner of the screen; all coordinates are expressed in pixels.

See Also SF_GetControlBottom, SF_PointInControl, SF_GetControlRight, SF_GetControlTop

SF_GetControlNextControl

CONTROL_REC * SF_GetControlNextControl (CONTROL_REC **pControl*)

Returns the next control of a form relative to the specified control.

Parameter *pControl* Pointer to the desired control.

Return Value Pointer to the next control of the form if successful; NULL if the function fails.

Comments This function fails if there is no next control in the form. The next control in a form is the internally stored next control and not necessarily the next control the user sees.

Use this function with `SF_GetFormFirstControl` to iterate through all the controls on all the pages of a form.

See Also SF_GetFirstForm, SF_GetFormFirstControl, SF_GetFormNextForm

SF_GetControlOsIndex

WORD SF_GetControlOsIndex (CONTROL_REC **pControl*)

Returns the OS index of the specified Satellite Forms control.

Parameter *pControl* Pointer to the desired control.

Return Value Index of the specified control in the OS form's control array if successful; 0xFFFF if the function fails.

Comments Pointers to controls are stored by the OS in an array in the OS form. This function returns the index in that array.

SF_GetControlPageNum

WORD SF_GetControlPageNum (CONTROL_REC **pControl*)

Returns the page number on which a control is located.

Parameter *pControl* Pointer to the desired control

Return Value Zero-based page number where the control is located.

SF_GetControlRight

WORD SF_GetControlRight (CONTROL_REC **pControl*)

Returns the right coordinate of a control.

Parameter *pControl* Pointer to the desired control.

Return Value *x* coordinate of the right side of the control's rectangle.

Comments The origin is the upper-left corner of the screen; all coordinates are expressed in pixels.

See Also SF_GetControlBottom,SF_GetControlLeft, SF_PointInControl, SF_GetControlTop

SF_GetControlTop

WORD SF_GetControlTop (CONTROL_REC **pControl*)

Returns the top coordinate of a control.

Parameter *pControl* Pointer to a control.

Return Value *y* coordinate of the top of the control's rectangle.

Comments The origin is the upper-left corner of the screen; all coordinates are expressed in pixels.

See Also SF_GetControlBottom,SF_GetControlLeft, SF_PointInControl, SF_GetControlRight

SF_GetControlType

WORD SF_GetControlType (CONTROL_REC **pControl*)

Returns the type of a control.

Parameter *pControl* Pointer to the desired control.

Return Value Type of control.

Comments The following control types are defined in SFDefs.h:

`CTRLTYPE_DROPLIST`: Drop List

`CTRLTYPE_TEXT`: Text

`CTRLTYPE_TITLE`: Title

`CTRLTYPE_FIELD`: Edit Control

`CTRLTYPE_BITMAP`: Bitmap

`CTRLTYPE_BUTTON`: Button

`CTRLTYPE_PARAGRAPH`: Paragraph

`CTRLTYPE_LIST`: List

`CTRLTYPE_CHECKBOX`: Check Box

`CTRLTYPE_RADIO`: Radio Button

`CTRLTYPE_INK`: Ink

`CTRLTYPE_LOOKUP`: Lookup

`CTRLTYPE_AUTOSTAMP`: Auto Stamp

`CTRLTYPE_GRAFFITI`: Graffiti Shift Indicator

`CTRLTYPE_CUSTOM`: Custom Control

SF_GetCreatorId

SF_GetCreatorId(???)

Parameters

Return Value

Comments

See Also

SF_GetCurrentForm

FORM_HEADER * SF_GetCurrentForm ()

Returns the current form.

Parameters None

Return Value Pointer to the current form.

Comments This function returns the current Satellite Forms form. Do not confuse this with the OS form. To get the OS form, use `SF_GetFormOsFormPtr`.

See Also SF_GetFormOsFormPtr

SF_GetEngineVersion

DWORD SF_GetEngineVersion ()

Retrieves the full version number of the Satellite Forms engine.

Parameters None

Return Value A DWORD value representing the version of the Satellite Forms engine.

Comments The version number is encoded in hexadecimal as *00MMmmRR*, where *MM* represents the major version, *mm* the minor version, and *RR* the revision number. The engine typically displays this number in its About box as MM.mm.RR. For example, 3.0.2.

SF_GetExtensionControl

CONTROL_REC * SF_GetExtensionControl ()

Returns a pointer to the Satellite Forms control associated with an SFX Custom control.

Parameters None

Return Value Pointer to a Satellite Forms control.

Comments All SFX Custom controls are associated with Satellite Forms controls.

SF_GetFieldCopy

DB_ITEM * SF_GetFieldCopy (TABLE_REC **pRec,* WORD *RowNum,* WORD *ColIndex*)

Retrieves a copy of the contents of the specified field.

Parameters

pRec Pointer to the table containing the field whose contents are to be retrieved.

RowNum Number of the row containing the data to be retrieved.

ColIndex Index of the column containing the data.

Return Value A pointer to the allocated copy of the data in the specified field; NULL if the function fails.

Comments This function is different from the similar function `SF_GetCachedField` in certain important respects. The data returned from this function is always on the dynamic heap so it can be readily modified. This also means that when you are done with the date, you must free it by calling `SF_xfree` to avoid causing a memory leak.

Another difference is that you do not need to cache a record explicitly with `SF_LockRecordAndCache` before using this function.

Caution: If the record being referenced is currently cached, any changes you make may be overwritten when the cache is committed.

SF_GetFieldOffset

WORD SF_GetFieldOffset (BYTE **pRecord,* WORD *ColIndex*)

Returns the offset from the beginning of a record to a particular field.

Parameters

pRecord Pointer to a locked memory block containing the desired Satellite Forms table record.

ColIndex Index of the column that represents the desired field.

Return Value The offset, in bytes, to the specified field; xFFFF if the function fails.

Comments This function is used internally by the engine to find fields within a record. This function cannot be used with cached records.

SF_GetFirstField(Obsolete. Do not use)

WORD SF_GetFirstField *(*TABLE_REC **pRec,* WORD *RowNum)*

Returns the OS record for the first field in the specified record. This is an internal system function and is not normally used.

Parameters *pRec* Pointer to the desired table.

RowNum Zero-based row number of the desired record.

Return Value OS record number of the first field of the specified row.

Comments **Caution:** This function is dependent on the Satellite Forms internal data structures and may change in the future.

SF_GetFirstForm

FORM_HEADER * SF_GetFirstForm ()

Returns the first form in the application.

Parameters None

Return Value Pointer to the first form in the application

Comments The first form in the application is the internally stored first form and not necessarily the initial form that the user sees when the application starts. Use this function with `SF_GetFormNextForm` when you need to iterate through all the forms in an application.

See Also SF_GetControlNextControl, SF_GetFormFirstControl, SF_GetFormNextForm

SF_GetFocusObjectPtr

FieldPtr SF_GetFocusObjectPtr ()

Returns the OS field object that has the current focus.

Parameters None

Return Value Pointer to the underlying OS field that has the current focus if successful; NULL if the function fails.

Comments This function fails if no object has focus.

SF_GetFormCurrentRow

WORD SF_GetFormCurrentRow (FORM_HEADER **pForm*)

Returns the row number of the current record in a form.

Parameter *pForm* Pointer to the desired form.

Return Value Zero-based row number of the current record if successful; 0xFFFF if the function fails.

Comments This function fails if there is no current row, that is, if the form is not linked to a table or if no rows meet the criteria of all active filters.

See Also SF_GetFormCurrPage

SF_GetFormCurrPage

WORD SF_GetFormCurrPage (FORM_HEADER **pForm*)

Returns the page number of the current page on the current form.

Parameter *pForm* Pointer to the current form.

Return Value Zero-based page number of the current page on the current form

See Also SF_GetFormCurrentRow

SF_GetFormFirstControl

CONTROL_REC * SF_GetFormFirstControl (FORM_HEADER **pForm*)

Returns a pointer to the first control on a form.

Parameter *pForm* Pointer to the desired form.

Return Value Pointer to the first control of the form if successful; NULL if the function fails.

Comments This function fails if there are no controls on the specified form. The first control of the application is the internally stored first control and not necessarily the first control that the user sees.

Use this function with `SF_GetControlNextControl` when you need to iterate through all the controls on a form.

See Also SF_GetControlNextControl, SF_GetFirstForm, SF_GetFormNextForm

SF_GetFormFlags

WORD SF_GetFormFlags (FORM_HEADER **pForm*)

Returns the attribute flags for a form.

Parameter *pForm* Pointer to the desired form.

Return Value Form attribute flags.

Comments The following form flags are defined in SFDefs.h. These flags set the user permissions for the form.

`FFLAG_ALLOWCREATE`: Create Record

`FFLAG_ALLOWDELETE`: Delete Record

`FFLAG_ALLOWMODIFY`: Modify

`FFLAG_ALLOWNAV`: Navigate

`FFLAG_ALLOWDELETELAST`: Delete Last Record

SF_GetFormNextForm

FORM_HEADER * SF_GetFormNextForm (FORM_HEADER **pForm*)

Returns a pointer to the next form in an application relative to the specified form.

Parameter *pForm* Pointer to the referenced form.

Return Value Pointer to the next form of the application if successful; NULL if the function fails.

Comments This function fails if there is no next form in the application. The next form of an application is the internally stored next form and not necessarily the next form the user sees.

Use this function with `SF_GetFirstForm` to iterate through all the forms in an application.

See Also SF_GetControlNextControl, SF_GetFirstForm, SF_GetFormFirstControl

SF_GetFormNumPages

WORD SF_GetFormNumPages (FORM_HEADER **pForm*)

Returns the number of pages in a form.

Parameter *pForm* Pointer to the desired form.

Return Value Number of pages in the form.

SF_GetFormOsFormPtr

FormPtr SF_GetFormOsFormPtr (FORM_HEADER **pForm*)

Returns the OS form that corresponds to a Satellite Forms form.

Parameter *pForm* Pointer to the desired Satellite Forms form

Return Value Pointer to the OS form that corresponds to the Satellite Forms form.

Comments This function returns the OS form. Do not confuse it with the Satellite Forms form. To get a Satellite Forms form, use `SF_GetCurrentForm`.

See Also SF_GetCurrentForm

SF_GetFormReturnIndex

WORD SF_GetFormReturnIndex (FORM_HEADER **pForm*)

Returns the index of the form that called a specified form.

Parameter *pForm* Pointer to the desired form.

Return Value Index of the calling form if successful; 0xFFFF if the function fails.

Comments Use this function to determine the form that called a specified form. This is useful when you want to enable the user to return to the calling form after completing an operation in the called form.

SF_GetFormTableIndex

WORD SF_GetFormTableIndex (FORM_HEADER **pForm*)

Returns the index of a form's linked table.

Parameter *pForm* Pointer to the desired form.

Return Value Index of the form's linked table if successful;0xFFFF if the function fails.

Comments This function fails if there is no table linked to the specified form.

SF_GetGlobalPtr

void * SF_GetGlobalPtr ()

Returns a pointer to the global data area used by an SFX Custom control.

Parameters None

Return Value Pointer to the global data area used by an SFX Custom control.

Comments The global data area is common to all instances of one type of SFX Custom control.

See Also SF_SetGlobalPtr, SF_GetInstanceDataPtr, SF_SetInstanceDataPtr

SF_GetInstanceDataPtr

void * SF_GetInstanceDataPtr ()

Returns a pointer to the instance data area used by an SFX Custom control.

Parameters None

Return Value Pointer to the instance data area used by an SFX Custom control.

Comments The instance data area is specific to a single instance of an SFX Custom control. Each instance of a control shares a common global data area while maintaining its own instance data area.

Set the pointer to the instance data area with `SF_SetInstanceDataPtr`.

See Also SF_GetGlobalPtr, SF_SetGlobalPtr, SF_SetInstanceDataPtr

SF_GetNumRows

WORD SF_GetNumRows (TABLE_REC **pRec*)

Returns the number of records in a table (affected by filters).

Parameter *pRec* Pointer to the desired table.

Return Value Number of records in the table.

Comments If a table has no records or if no records meet the criteria of all active filters, this function returns zero.

SF_GetRowItemCopy(Obsolete. Do not use.)

void * SF_GetRowItemCopy (TABLE_REC **pRec,* WORD *RowNum,* WORD *ColIndex,* WORD **pSize*)

Copies data from a record.

Parameters *pRec* Pointer to the desired table.

RowNum Zero-based row number of record from which data is to be copied.

ColIndex Index of column from which data is to be copied.

pSize Pointer to a WORD that receives the size of the data, not including the terminating zero.

Return Value Copy of data from the specified table, row, and column if successful; NULL if the function fails.

Comments Set the input parameter *pSize* to NULL if you are not interested in the size of the data. This function allocates memory with `SF_xmalloc`. Free this memory with `SF_xfree`.

See Also SF_CommitItemToRow (Obsolete. Do not use.), SF_ConvertInternalToDisplayFormat

SF_GetSysTime

DWORD SF_GetSysTime ()

Returns the number of seconds since 00:00:00, January 1, 1904.

Parameters None

Return Value Seconds since 00:00:00, January 1, 1904.

SF_GetTableColNumDecimals

WORD SF_GetTableColNumDecimals (TABLE_REC **pRec,* WORD *ColIndex*)

Returns the number of decimal places in a numeric column.

Parameters *pRec* Pointer to the desired table.

ColIndex Index of the desired numeric column.

Return Value The number of decimal places in the column.

See Also SF_GetTableColType, SF_GetTableColWidth

SF_GetTableColType

WORD SF_GetTableColType (TABLE_REC **pRec,* WORD *ColIndex*)

Returns a column’s type.

Parameters *pRec* Pointer to the desired table.

ColIndex Index of the desired column.

Return Value The column’s type.

Comments The following column types are defined in SFDefs.h.

`PDA_FIELDTYPE_CHAR`: Character

`PDA_FIELDTYPE_DATE`: Date

`PDA_FIELDTYPE_TIME`: Time

`PDA_FIELDTYPE_INTEGER`: Integer

`PDA_FIELDTYPE_FLOAT`: Float

`PDA_FIELDTYPE_LOGICAL`: True/False

`PDA_FIELDTYPE_INK`: Ink

`PDA_FIELDTYPE_INT64`: 64-bit Integer

`PDA_FIELDTYPE_TIMESTAMP`: Time Stamp

`PDA_FIELDTYPE_ERROR`: Error Condition

See Also SF_GetTableColNumDecimals, SF_GetTableColWidth

SF_GetTableColWidth

WORD SF_GetTableColWidth (TABLE_REC **pRec,* WORD *ColIndex*)

Returns the maximum width of a column.

Parameters	*pRec*	Pointer to the desired table.
	ColIndex	Index of the desired column.

Return Value The maximum width of the column in characters.

See Also SF_GetTableColNumDecimals, SF_GetTableColType

SF_GetUIObjectParent

CONTROL_REC * SF_GetUIObjectParent (void **pObject*)

Returns a pointer to the Satellite Forms control associated with a specified OS control.

Parameter	*pObject*	Pointer to OS control.

Return Value Pointer to a Satellite Forms control if successful; NULL if the function fails.

Comments Only call this function for OS controls the Satellite Forms engine creates, not for controls that you create yourself with OS API.

See Also SF_OsIndexToControlRec

SF_GotoNewForm

CBOOL SF_GotoNewForm (WORD *NewFormIndex,* WORD *JumpOpt,* CBOOL *fSaveRetIndex*)

Jumps to a new form. The new form becomes the current form.

Parameters	*NewFormIndex*	Index of the new form.
	JumpOpt	Record creation options.
	fSaveRetIndex	Flag that specifies whether to save the index of the calling form.

Return Value TRUE if the jump is successful; FALSE if not.

Comments This function fails if the jump option **Fail if no records** is set and there are no records in the target form that satisfy all active filters.

The following record creation types associated with the jump are defined in SFDefs.h.

`JUMPOPT_CREATE_IF_NONE`: Create if no records

`JUMPOPT_ALLOW_IF_NONE`:Allow if no records

`JUMPOPT_FAIL_IF_NONE`:Fail if no records

`JUMP_OPT_ALWAYS_CREATE`: Always create record

SF_GotoRow

void SF_GotoRow (WORD *NewRow,* CBOOL *fSaveCurrent,* CBOOL *fNotifyRecCreate*)

Makes the specified record the current record.

Parameters	*NewRow*	Zero-based row number of the record that becomes the current record.
	fSaveCurrent	Flag that specifies whether to save the contents of the controls to the current record before switching records.
	fNotifyRecCreate	Flag that specifies whether to fire the form event `AfterRecCreate` after the move to the new record

Return Value None

Comments Set the flag *fSaveCurrent* to TRUE unless the current record has been deleted.

SF_IdToControlRec

CONTROL_REC * SF_IdToControlRec (WORD *ControlId*)

Converts an OS control ID to a pointer to the OS control.

Parameter *ControlId* OS control ID.

Return Value Pointer to the corresponding OS control if successful; NULL if the function fails.

SF_IdToObjectPtr

void * SF_IdToObjectPtr (WORD *ObjectId*)

Converts an OS object ID to a pointer to the Satellite Forms object.

Parameter *ObjectId* OS Object ID.

Return Value Pointer to the Satellite Forms object if successful; NULL if the function fails.

SF_InfoMsg

void SF_InfoMsg (char **p*)

Displays a dialog box containing a message and an OK button.

Parameter *p* Pointer to a string containing the message displayed in the dialog box.

Return Value None

See Also SF_ConfirmMsg, SF_InfoMsgInt

SF_InfoMsgInt

void SF_InfoMsgInt (char **p*, DWORD *d*)

Displays a dialog box containing a message followed by an integer.

Parameters	*p*	Pointer to a string containing the message displayed in the dialog box.
	d	Integer that is concatenated to the message.

Return Value None

See Also SF_ConfirmMsg, SF_InfoMsg

SF_InstallHandler

void SF_InstallHandler (int *EventIndex,* CBOOL **pHandler*)

Installs an event handler.

Parameters	*EventIndex*	Event to be handled.
	pHandler	Pointer to the event handler function.

Return Value None

Comments Note that the `AfterOpen` and `BeforeClose` events in the API differ from the corresponding events in the scripting language. In the scripting language, these events fire one time per form, upon entry and exit of the form. In the API, they fire one time per page, upon entry and exit of the page.

The following event types are defined in SFDefs.h:

`SF_Handler_Form_AfterOpen`: Occurs after a form or a form's page is opened.

`SF_Handler_Form_AfterLoad`: Occurs after the controls on a form are loaded with data from the form's table

`SF_Handler_Form_AfterChange`: Occurs after data in any field on the form changes.

`SF_Handler_Form_AfterRecCreate`: Occurs after a new record is created in a form.

`SF_Handler_Form_BeforeClose`: Occurs before a form or page is closed (exited).

`SF_Handler_Form_BeforeRecDelete`: Occurs before a record is deleted from a form.

`SF_Handler_Form_Onvalidate`: Occurs when a form is validated.

`SF_Handler_OnExtLoad`: Occurs when the application is opened and the extension is loaded.

`SF_Handler_OnExtUnload`: Occurs when the application is closed and the extension is unloaded.

`SF_Handler_OnFormEvent`: Occurs when the OS issues a form event.

`SF_Handler_OnSysEvent`: Occurs when the OS issues a system event.

`SF_Handler_OnSfxNotify`: Occurs when a user accesses one of the methods of an SFX Custom control or the system notifies an SFX Custom control.

SF_InternalDateToSysDate

long SF_InternalDateToSysDate (char **pInternal*)

Converts the Satellite Forms internal date representation in YYYYMMDD format to days since January 1, 1904.

Parameter	*pInternal*	Pointer to a string containing a date in Satellite Forms internal date representation.

Return Value Days since January 1, 1904.

See Also SF_InternalTimeToSysTime, SF_GetSysTime, SF_SysDateToInternalDate, SF_SysTimeToInternalTime

SF_InternalTimeToSysTime

long SF_InternalTimeToSysTime (char **pInternal*)

Converts the Satellite Forms internal time representation in 24-hour HH:MM:SS format to seconds since midnight.

Parameter *pInternal* Pointer to a string containing a time in Satellite Forms internal time representation.

Return Value Seconds since midnight.

See Also SF_InternalDateToSysDate, SF_GetSysTime, SF_SysDateToInternalDate, SF_SysTimeToInternalTime

SF_InternalToPilotDate

void SF_InternalToPilotDate (char **pPilotDate*, DB_ITEM **pInternal*)

Converts the Satellite Forms internal representation of date to the OS representation.

Parameters *pPilotDate* Pointer to a buffer that receives the OS date representation.

pInternal Pointer to a zero-terminated string in the internal Satellite Forms date format.

Return Value None

Comments The Satellite Forms internal date representation is in year 2000-compliant YYYYMMDD format. The country–specific OS date representation is set in the handheld device's preferences. The buffer that receives the handheld device date representation must be at least 11 bytes in length.

See Also SF_InternalToPilotTime, SF_PilotDateToInternal, SF_PilotTimeToInternal, SF_InternalDateToSysDate, SF_InternalTimeToSysTime, SF_SysDateToInternalDate, SF_SysTimeToInternalTime

SF_InternalToPilotTime

void SF_InternalToPilotTime (char **pPilotTime*, DB_ITEM **pInternal*)

Converts the Satellite Forms internal representation of time to the OS representation.

Parameters *pPilotTime* Pointer to a buffer that receives the handheld device time representation.

pInternal Pointer to a zero-terminated string in the internal Satellite Forms time format.

Return Value None

Comments The Satellite Forms internal time representation is in 24-hour HH:MM:SS format. The country–specific OS time representation is set in the handheld device's preferences. The buffer that receives the handheld device time representation must be at least 12 bytes in length.

See Also SF_InternalToPilotDate, SF_PilotDateToInternal, SF_PilotTimeToInternal, SF_InternalDateToSysDate, SF_InternalTimeToSysTime, SF_SysDateToInternalDate, SF_SysTimeToInternalTime

SF_itoa

char * SF_itoa (const char **pszStr*, long *i*)

Converts an integer to a string.

Parameter *pszStr* Pointer to the string where the result is stored.

i Integer to convert.

Return Value Pointer to the converted string.

SF_itoh

char * SF_itoh (char **pszStr*, DWORD *i*)

Converts an integer to a hexadecimal ASCII string.

Parameter	*pszStr*	Pointer to the string where the result is stored.
	i	Integer to convert.
Return Value	Pointer to the converted string.	

SF_LoadCtrlObjFromCachedRecord

void SF_LoadCtrlObjFromCachedRecord (FormPtr *frm*, CONTROL_REC **pControl*, CACHED_REC **pCachedRec*)

Loads a Check Box or Radio Button control with data from a cached record. Replaces `SF_LoadCtrlObjFromRow`.

Parameters	*frm*	Pointer to the desired OS form.
	pControl	Pointer to the Check Box or Radio Button control that receives the data.
	pCachedRec	Pointer to the desired cached record.
Return Value	None	
Comments	The control knows the column of the table to which it is bound.	
See Also	SF_LoadDropListFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_LoadInkFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveDropListToCachedRecord, SF_SaveFieldToCachedRecord, SF_SaveInkFieldToCachedRecord	

SF_LoadDropListFromCachedRecord

void SF_LoadDropListFromCachedRecord (FormPtr *frm*, CONTROL_REC **pControl*, CACHED_RECORD **pCachedRec*)

Loads a Drop List control with data from a cached record. Replaces `SF_LoadDropListFromRow`.

Parameters	*frm*	Pointer to the desired OS form.
	pControl	Pointer to the Drop List control that receives the data.
	pCachedRec	Pointer to the desired cached record.
Return Value	None	
Comments	The control knows the column of the table to which it is bound. The control also knows the associated lookup table and performs a lookup.	
See Also	SF_LoadCtrlObjFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_LoadInkFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveDropListToCachedRecord, SF_SaveFieldToCachedRecord, SF_SaveInkFieldToCachedRecord	

SF_LoadFieldFromCachedRecord

void SF_LoadFieldFromCachedRecord (FormPtr *frm*, CONTROL_REC **pControl*, CACHED_RECORD **pCachedRec*)

Loads an edit control with data from a cached record. Replaces `SF_LoadFieldFromRow`.

Parameters	*frm*	Pointer to the desired OS form.
	pControl	Pointer to the Edit control that receives the data.
	pCachedRec	Pointer to the desired cached record.
Return Value	None	

Comments The control knows the column of the table to which it is bound.

See Also SF_LoadCtrlObjFromCachedRecord, SF_LoadDropListFromCachedRecord, SF_LoadInkFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveDropListToCachedRecord, SF_SaveFieldToCachedRecord, SF_SaveInkFieldToCachedRecord

SF_LoadFormWithCurrentRow

void SF_LoadFormWithCurrentRow ()

Loads the controls on the current page with data from the current record of the form's linked table.

Parameters None

Return Value None

Comments If there is no current row, this function does nothing.

SF_LoadInkFieldFromCachedRecord

void SF_LoadInkFieldFromCachedRecord (FormPtr *frm*, CONTROL_REC **pControl,* CACHED_RECORD **pCachedRec*)

Loads an Ink control with data from a cached record. Replaces `SF_LoadInkFieldFromRow`.

Parameters *frm* Pointer to the desired OS form.

pControl Pointer to the Ink control that receives the data.

pCachedRec Pointer to the desired cached record

Return Value None

Comments The control knows the column of the table to which it is bound.

See Also SF_LoadCtrlObjFromCachedRecord, SF_LoadDropListFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveDropListToCachedRecord, SF_SaveFieldToCachedRecord, SF_SaveInkFieldToCachedRecord

SF_LockRecordAndCache

CACHED_RECORD * SF_LockRecordAndCache (TABLE_REC **pRec*, WORD *RowNum*)

Locks the specified record and caches it in dynamic memory.

Parameters *pRec* Pointer to the desired table.

RowNum Row number of record to cache.

Return Value A pointer to the newly allocated block of memory that contains the cached record.

Comments Call this function to cache a record into memory.

Since records comprise several compacted fields, the easiest way to modify a field is to cache the record, use `SF_GetCachedField` to modify it, and then use `SF_CommitCachedRecord` to save it back to its table.

If you do not commit a cached record, you must call `SF_FreeCachedRecordData` to dispose of its data. Failure to free the cached record results in a memory leak.

See Also SF_GetCachedField, SF_SetCachedField, SF_FreeCachedRecordData

SF_LockRowItem(Obsolete. Do not use)

Void * SF_LockRowItem (TABLE_REC **pRec*, WORD *RowNum*, WORD *ColIndex*, WORD **pSize*)

Locks a value in memory. This is an internal function and not normally used.

Parameters *pRec* Pointer to the desired table.

RowNum Zero-based row number of record containing the data.

ColIndex Index of column containing the data.

pSize Pointer to a WORD that receives the size of the value, not including the terminating zero.

Return Value Pointer to the actual value in database memory.

Comments This value is read-only because database memory is read-only.

See Also SF_GetRowItemCopy (Obsolete. Do not use.), SF_ResizeLockedRecord (Obsolete. Do not use.), SF_UnlockRowItem (Obsolete. Do not use)

SF_memcpy

SF_memcpy(??? ???)

Copies one block of memory into another.

Parameters *???* ???

??? ???

Return Value ???

SF_memset

Err SF_memset (void **pDest*, long *numBytes*, BYTE *value*)

Fills a range of dynamic memory range with the specified value.

Parameters *pDest* Pointer to the block of memory to fill.

njmBytes Number of bytes to fill.

value Value to be filled into the memory range.

Return Value Returns 0 if successful; error code if the function fails.

SF_memsize

DWORD SF_memsize (const void **p*)

Returns the size of the specified pointer.

Parameters *p* The desired pointer.

Return Value Returns the size of the specified pointer.

SF_OsIndexToControlRec

CONTROL_REC * SF_OsIndexToControlRec (WORD *ObjIndex*)

Converts an index in the OS form's control array to a pointer to the corresponding Satellite Forms control.

Parameter *ObjIndex* Index of the desired control in the OS control array.

Return Value Pointer to the Satellite Forms control if successful; NULL if the function fails.

Comments Only call this function for OS controls the Satellite Forms engine creates, not for controls that you create yourself with OS API.

See Also SF_GetUIObjectParent

SF_PerformControlAction

CBOOL SF_PerformControlAction (CONTROL_REC **pControl*)

Executes the action associated with a control.

Parameter *pControl* Pointer to a control.

Return Value TRUE if the action is executed successfully; FALSE if it failed to execute successfully.

See Also SF_GetControlAction

SF_PilotDateToInternal

CBOOL SF_PilotDateToInternal (DB_ITEM **pInternal*, char **pPilotDate*)

Converts the OS representation of date to the Satellite Forms internal representation.

Parameters *pInternal* Pointer to a buffer that receives the internal date representation.

pPilotDate Pointer to a zero-terminated string in the handheld device date format.

Return Value TRUE if the conversion is successful; FALSE if the function fails.

Comments The Satellite Forms internal date representation is in year 2000-compliant YYYYMMDD format. The country–specific OS time representation is set in the handheld device’s preferences. The buffer that receives the internal date representation must be at least 9 bytes in length.

See Also SF_InternalToPilotDate, SF_InternalToPilotTime, SF_PilotTimeToInternal, SF_InternalDateToSysDate, SF_InternalTimeToSysTime, SF_SysDateToInternalDate, SF_SysTimeToInternalTime

SF_PilotTimeToInternal

CBOOL SF_PilotTimeToInternal (DB_ITEM **pInternal*, char **pPilotTime*)

Converts the OS representation of time to the Satellite Forms internal representation.

Parameters *pInternal* Pointer to a buffer that receives the internal time representation.

pPilotTime Pointer to a zero-terminated string in the handheld device time format.

Return Value TRUE if the conversion is successful; FALSE the function fails.

Comments The Satellite Forms internal time representation is in 24-hour HH:MM:SS format. The country–specific OS time representation is set in the handheld device’s preferences. The buffer that receives the internal time representation must be at least 9 bytes in length.

See Also SF_InternalToPilotDate, SF_InternalToPilotTime, SF_PilotDateToInternal, SF_InternalDateToSysDate, SF_InternalTimeToSysTime, SF_SysDateToInternalDate, SF_SysTimeToInternalTime

SF_PointInControl

CBOOL SF_PointInControl (CONTROL_REC **pControl*, WORD *x*, WORD *y*)

Determines if a point, identified by an (x,y) coordinate, lies within a control’s rectangle.

Parameters *pControl* Pointer to the desired control.

x *x* coordinate of a point

y *y* coordinate of a point

Return Value TRUE if the (x,y) coordinate lies within the control’s rectangle; FALSE if the (x,y) coordinate lies outside the control’s rectangle

Comments The origin is the upper-left corner of the screen; all coordinates are expressed in pixels.

See Also SF_GetControlBottom,SF_GetControlLeft, SF_GetControlRight, SF_GetControlTop

SF_PointToControlRec

CONTROL_REC * SF_PointToControlRec (WORD *xPos*, WORD *yPos*, WORD *CtrlType*)

Converts a point, identified by an (x,y) coordinate, into a pointer to a control that is located at that position.

Parameters

Parameters	*xPos*	x coordinate of the desired point.
	yPos	y coordinate of the desired point.
	CtrlType	Type of control, or zero.

Return Value Pointer to the control if successful; NULL if there is no control at the desired coordinates.

Comments Only controls of type *CtrlType* are checked To check all controls, pass zero as *CtrlType*. The following control types are defined in SFDefs.h:

`CTRLTYPE_DROPLIST`: Drop List

`CTRLTYPE_TEXT`: Text

`CTRLTYPE_TITLE`: Title

`CTRLTYPE_EDIT`: Edit Control

`CTRLTYPE_BITMAP`: Bitmap

`CTRLTYPE_BUTTON`: Button

`CTRLTYPE_PARAGRAPH`: Paragraph

`CTRLTYPE_LIST`: List Box

`CTRLTYPE_CHECKBOX`: Check Box

`CTRLTYPE_RADIO`: Radio Button

`CTRLTYPE_INK`: Ink

`CTRLTYPE_LOOKUP`: Lookup

`CTRLTYPE_AUTOSTAMP`: Auto Stamp

`CTRLTYPE_GRAFFITI`: Graffiti Shift Indicator

`CTRLTYPE_CUSTOM`: Custom Control

SF_QueryField

DB_ITEM * SF_QueryField (TABLE_REC **pRec*, WORD *RowNum*, WORD *ColIndex*)

Provides a fast way to obtain a read-only pointer to a field.

Parameters	*pRec*	Pointer to the desired table.
	RowNum	Row number of record to be accessed.
	ColIndex	Index of the desired field.

Return Value Read-only pointer to the specified field; NULL if an error occurs.

Comments If you do not need to modify the data, this function provides a fast alternative to `SF_GetFieldCopy` or `SF_GetCachedField`.

Caution: You must call `SF_UnqueryField` to release a field accessed with this function. Failure to do so leaves records locked in the table.

See Also SF_UnqueryField

SF_RenderInk

void SF_RenderInk (CONTROL_REC **pControl*, TABLE_REC **pRec*, WORD *RowNum*)

Redraws the contents of an Ink control.

Parameters	*pControl*	Pointer to the desired ink control.
	pRec	Pointer to the desired table.
	RowNum	Zero-based row number of the desired record in the table.
Return Value	None	
Comments	The control knows the column to which it is bound.	

SF_ResizeLockedRecord(Obsolete. Do not use.)

void * SF_ResizeLockedRecord (TABLE_REC **pRec*, WORD *RowNum*, WORD *ColIndex*)

Resizes a value locked with `SF_LockRowItem`. This is an internal system function and not normally used.

Parameters	*pRec*	Pointer to the desired table.
	RowNum	Zero-based row number of record containing the data.
	ColIndex	Index of column containing the data.
Return Value	Pointer to the resized value in database memory.	
Comments	The location of the value may move as a result of the resizing.	
See Also	SF_LockRowItem (Obsolete. Do not use), SF_UNLOCKROWITEM (OBSOLETE. DO NOT USE)	

SF_RowMeetsCriteria

CBOOL SF_RowMeetsCriteria (TABLE_REC **pRec*, WORD *RowNum*)

Determines whether a record meets the criteria of all active filters.

Parameters	*pRec*	Pointer to the desired table.
	RowNum	Zero-based row number of record to be tested.
Return Value	TRUE if the record meets the criteria of all active filters; FALSE if the record fails to meet some or all criteria.	

SF_SaveCtrlObjToCachedRecord

void SF_SaveCtrlObjToCachedRecord (CONTROL_REC **pControl*, CACHED_RECORD **pCachedRec*)

Saves the contents of a Check Box or Radio Button control to a cached record. Replaces `SF_SaveCtrlObjToRow`.

Parameters	*pControl*	Pointer to the desired Check Box or Radio Button control.
	pCachedRec	Pointer to the desired cached record.
Return Value	None	
Comments	The control knows the column of the table to which it is bound.	
See Also	SF_LoadCtrlObjFromCachedRecord, SF_LoadDropListFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_LoadInkFieldFromCachedRecord, , SF_SaveDropListToCachedRecord, SF_SaveFieldToCachedRecord, SF_SaveInkFieldToCachedRecord	

SF_SaveDropListToCachedRecord

void SF_SaveDropListToCachedRecord (CONTROL_REC **pControl*, CACHED_RECORD **pCachedRec*)

Saves the contents of a Drop List control to a cached record. Replaces `SF_SaveDropListToRow`.

Parameters *pControl* Pointer to the desired drop list control.

pCachedRec Pointer to the desired cached record.

Return Value None

Comments The control knows the column of the table to which it is bound.

See Also SF_LoadCtrlObjFromCachedRecord, SF_LoadDropListFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_LoadInkFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveFieldToCachedRecord, SF_SaveInkFieldToCachedRecord

SF_SaveFieldToCachedRecord

void SF_SaveFieldToCachedRecord (CONTROL_REC **pControl*, CACHED_RECORD **pCachedRec*)

Saves the contents of an Edit control to a cached record. Replaces `SF_SaveFieldToRow`.

Parameters *pControl* Pointer to the desired edit control.

pCachedRec Pointer to the desired cached record.

Return Value None

Comments The control knows the column of the table to which it is bound.

See Also SF_LoadCtrlObjFromCachedRecord, SF_LoadDropListFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_LoadInkFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveDropListToCachedRecord, SF_SaveInkFieldToCachedRecord

SF_SaveInkFieldToCachedRecord

void SF_SaveInkFieldToCachedRecord (CONTROL_REC **pControl*, CACHED_RECORD **pCachedRec*)

Saves the contents of an Ink control to a cached record. Replaces `SF_SaveInkFieldToRow`.

Parameters *pControl* Pointer to the desired ink control.

pCachedRec Pointer to the desired cached record.

Return Value None

Comments The control knows the column of the table to which it is bound.

See Also SF_LoadCtrlObjFromCachedRecord, SF_LoadDropListFromCachedRecord, SF_LoadFieldFromCachedRecord, SF_LoadInkFieldFromCachedRecord, SF_SaveCtrlObjToCachedRecord, SF_SaveDropListToCachedRecord, SF_SaveFieldToCachedRecord

SF_ScriptExecExt

Void SF_ScriptExecExt (char *NumArgs*, int *ExtIndex*, int *CtrlIndex*, CBOOL *fIsFunc*)

Executes a method of an extension.

Parameters	*NumArgs*	Number of arguments pushed onto the stack.
	ExtIndex	Index of the desired extension.
	CtrlIndex	Index of the desired control.
	fIsFunc	Flag that specifies if the extension is a function – if it returns a value on the stack.

Return Value Stack Result. Whatever the extension pushes onto the evaluation stack if the extension is a function; if the extension is not a function, there is no return value.

Comments The return value is pushed on the interpreter's evaluation stack, not the processor stack. To access the evaluation stack, use the `SF_Push*` and `SF_Pop*` functions.

SF_ScriptFloatToString

void SF_ScriptFloatToString (FlpDouble *FloatVal*, char **pBuff*, BYTE *BuffSize*, char *NumDec*)

Converts a floating-point value into a string.

Parameters	*FloatVal*	Floating-point value to be converted.
	pBuff	Pointer to a buffer that receives the formatted value.
	BuffSize	Size of the buffer.
	NumDec	Number of decimal places to use in formatting.

Return Value None

Comments Pass `-1` in *NumDec* to use the minimum possible number of decimal places. Trailing zeros and the decimal point are omitted if not necessary.

SF_ScriptFreeParamMem

void SF_ScriptFreeParamMem (STACK_ITEM **pParam*)

Frees memory associated with a parameter. This is an internal function and should not normally be used.

Parameter	*pParam*	Pointer to an item on the stack.

Return Value None

Comments Normally, the `SF_Pop*` functions free memory, making it unnecessary to use this function.

SF_ScriptGetTosPtr

STACK_ITEM * SF_ScriptGetTosPtr ()

Returns a pointer to the item on the top of the stack.

Parameters None

Return Value Pointer to the item on the top of the stack.

Comments This is an `SF_Script` function. Access the stack using the `SF_Push*` and `SF_Pop*` functions.

SF_ScriptPopFloat

flpDouble SF_ScriptPopFloat ()

Removes an item from the top of the stack and returns it as a floating-point number.

Parameter	*Data*	Stack Parameter. Item on the top of the stack.

Return Value Floating-point number converted from the top of the stack.

Comments If the item at the top of the stack not a floating-point number, it is converted into one.

Use the `SF_ScriptPop*` and `SF_ScriptPush*` functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPopInt64

SF_INT64 SF_ScriptPopInt64 ()

Removes an item from the top of the stack and returns it as a 64-bit integer.

Parameter *Data* Stack Parameter. Item on the top of the stack.

Return Value 64-bit integer converted from the top of the stack.

Comments If the item at the top of the stack is not an integer, it is converted into one.

Use the `SF_ScriptPop*` and `SF_ScriptPush*` functions to obtain parameters passed in by script invocations of an extension’s methods and to push return values back on the stack, respectively.

SF_ScriptPopInteger

long SF_ScriptPopInteger ()

Removes an item from the top of the stack and returns it as an integer.

Parameter *Data* Stack Parameter. Item on the top of the stack

Return Value Integer converted from the top of the stack.

Comments If the item at the top of the stack is not an integer, it is converted into one.

Use the `SF_ScriptPop*` and `SF_ScriptPush*` functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPopString

char * SF_ScriptPopString (WORD *ReqLen*)

Removes an item from the top of the stack and returns it as a string.

Parameters *ReqLen* Size of memory allocated for the data.

Data Stack Parameter. Item on the top of the stack.

Return Value String converted from the top of the stack.

Comments If the item at the top of the stack not a string, it is converted into one.

Setting *ReqLen* to zero automatically sizes the allocated memory to the size of the string.

This function allocates memory with `SF_xmalloc`. Free the string with `SF_xfree` when you are done with it.

Use the `SF_ScriptPop*` and `SF_ScriptPush*` functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushFloat

void SF_ScriptPushFloat (FlpDouble *FloatVal*)

Pushes a floating-point value onto the stack.

Parameter *FloatVal* Floating-point value.

Return Value None

Comments Use the SF_ScriptPop* and SF_ScriptPush* functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushFloatFromStr

long SF_ScriptPushFloatFromStr (char **pString*)**

Pushes a floating-point value onto the stack from a string value.

Parameter *pString* Pointer to the desired string.

Return Value The length of the string including the terminating zero.

Comments Use the SF_ScriptPop* and SF_ScriptPush* functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushInt64

void SF_ScriptPushInt64 (SF_INT64 *Int64Val*)

Pushes a 64-bit integer onto the stack.

Parameter *Int64Val* 64-bit integer value.

Return Value None

Comments Use the SF_ScriptPop* and SF_ScriptPush* functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushInteger

void SF_ScriptPushInteger (long *IntVal*)

Pushes an integer onto the stack.

Parameter *IntVal* Integer value.

Return Value None

Comments Use the SF_ScriptPop* and SF_ScriptPush* functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushStaticStr

long SF_ScriptPushStaticStr (char **pString*)**

Pushes a string not allocated with SF_xmalloc onto the stack.

Parameter *PString* Pointer to the desired literal string.

Return Value The length of the string including the terminating zero.

Stack Result. String value.

Comments Use this function to push strings not allocated with SF_xmalloc that are literals in your code.

Use the SF_ScriptPop* and SF_ScriptPush* functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushVar

void SF_ScriptPushVar (int *VarIndex*)

Pushes a variable onto the stack.

Parameter *VarIndex* Index of the desired variable.

Return Value Stack Result. Value of variable.

Comments Use the `SF_ScriptPop*` and `SF_ScriptPush*` functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptPushVarString

char * SF_ScriptPushVarString (char **pString*)

Pushes a string allocated with `SF_xmalloc` onto the stack.

Parameter *pString* Pointer to the desired string.

Return Value Pointer to the string that is now owned by the Satellite Forms engine.

Stack Result. String value.

Comments Use this function to push strings allocated with `SF_xmalloc` onto the stack and pass ownership of the string to the Satellite Forms engine. The engine frees memory associated with the string, so you do not need to use `SF_xfree`.

Use the `SF_ScriptPop*` and `SF_ScriptPush*` functions to obtain parameters passed in by script invocations of an extension's methods and to push return values back on the stack, respectively.

SF_ScriptVarAssign

void SF_ScriptVarAssign (int *VarIndex*)

Assigns the value on the top of the stack to a variable.

Parameter *VarIndex* Index of the desired variable.

Return Value None

Comments In order to obtain the index to a particular variable, the script that invoked your extension must pass back the index.

SF_SearchTable

WORD SF_SearchTable (TABLE_REC **pRec*, DB_ITEM **pItem*, WORD *ColIndex*)

Finds data in a table.

Parameters	*pRec*	Pointer to the desired table.
	pItem	Data to be looked up.
	ColIndex	Index of column where data is to be looked up.

Return Value Zero-based row number of the first record containing the data if successful; 0xFFFF if the function does not find the data.

Comments This function fails if it does not find the data.

SF_SetCachedField

DB_ITEM * SF_SetCachedField (CACHED_RECORD **pCachedRec*, WORD *ColIndex*, void **pData*, WORD *DataLen*)

Replaces the contents of a field in a cached record.

Parameters	*pCachedRec*	Pointer to the desired cached record.
	ColIndex	Index of column where cached field is to be set.
	pData	Pointer to the block of memory that contains the data to be stored in the cached field.
	DataLen	Size of data pointed at by *pData.*

Return Value A pointer to a `DB_ITEM` structure containing the field’s data; NULL if the function fails.

Comments A copy of the data passed in is stored in the cached record; therefore, you need not keep the data pointed to by *pData* after the function returns.

See Also SF_AllocDbItem, SF_FreeCachedRecordData, SF_GetFieldCopy

SF_SetCtrlObjText

void SF_SetCtrlObjText (ControlPtr *pCtrlObj*, char **pText*)

Sets the caption of a Check Box or Radio Button control.

Parameters	*pCtrlObj*	Pointer to the desired Check Box or Radio Button control.
	pText	Text to which the control’s caption is to be set.
Return Value	None	

SF_SetDropListText

void SF_SetDropListText (CONTROL_REC **pControl*, char **pText*)

Sets the caption of a Drop List control's trigger control.

Parameters	*pControl*	Pointer to the desired Drop List control.
	pText	Text to which the Drop List control’s trigger control caption is to be set
Return Value	None	

SF_SetFieldText

void SF_SetFieldText (FieldPtr *pField*, char **pText*, WORD *Size*)

Sets the caption of an Edit control.

Parameters	*PField*	Pointer to the desired Edit control.
	pText	Text to which the Edit control’s caption is to be set.
	Size	Size of the caption.
Return Value	None	

SF_SetGlobalPtr

void SF_SetGlobalPtr (void **pGlobals*)

Saves a pointer to the global data area an SFX Custom uses.

Parameter	*pGlobals*	Pointer to the block of memory that contains the global data.
Return Value	None	

Comments Allocate the memory block for the global data area with `SF_db_xmalloc`.

The global data area is common to all instances of an SFX Custom control. Use `SF_SetInstanceDataPtr` to save a pointer to an instance data area for a single instance of an SFX Custom control.

See Also SF_GetGlobalPtr, SF_GetInstanceDataPtr, SF_SetInstanceDataPtr

SF_SetInstanceDataPtr

void SF_SetInstanceDataPtr (void **pData*)

Saves a pointer to the instance data area an SFX Custom control uses.

Parameter	PData	Pointer to the block of memory that contains the instance data.
Return Value	None	

Comments Allocate the memory block for the data area with `SF_db_xmalloc`.

The instance data area is specific to a single instance of an SFX Custom control. Each instance of an SFX Custom control shares a common global data area and maintains its own instance data area.

See Also SF_GetGlobalPtr, SF_GetInstanceDataPtr, SF_SetGlobalPtr

SF_SetLookupText

void SF_SetLookupText (CONTROL_REC **pControl*, char **pText*)

Sets the caption of a Lookup control.

Parameters *pControl* Pointer to the desired Lookup control.

pText Text to which the lookup control's caption is to be set.

Return Value None

SF_ShowAbout

void SF_ShowAbout ()

Displays the Satellite Forms About box.

Parameters None

Return Value None

SF_strcat

char * SF_strcat (char **pszDest*, const char **pszSource*)

Concatenates a string to another.

Parameters *pszDest* Pointer to the destination string.

pszSource Pointer to the source string.

Return Value Pointer to the destination string.

SF_strchr

char * SF_strchr (const char **pszStr*, WChar *chr*)

Searches a string for specified character.

Parameters *pszDest* Pointer to the destination string.

pszSource Pointer to the source string.

Return Value Pointer to the destination string.

SF_strcmp

char * SF_strcmp (const char **pszStr1*, const char **pszStr2*)

Case-sensitive comparison of two strings.

Parameters *pszStr1* Pointer to the first string.

pszStr2 Pointer to the second string.

Return Value Returns 0 if the strings are identical; > 0 if *pszStr1* is alphabetically greater than *pszStr2*; < 0 if if *pszStr1* is alphabetically less than *pszStr2*.

SF_strcpy

char * SF_strcpy (char **pszDest*, const char **pszSource*)

Copies one string to another.

Parameters	*pszDest*	Pointer to the destination string.
	pszSource	Pointer to the source string.
Return Value	Pointer to the destination string.	

SF_stricmp

char * SF_stricmp (const char **pszStr1*, const char **pszStr2*)

Case-insensitive comparison of two strings.

Parameters	*pszStr1*	Pointer to the first string.
	pszStr2	Pointer to the second string.
Return Value	Returns 0 if the strings are identical; > 0 if *pszStr1* is alphabetically greater than *pszStr2*; < 0 if if *pszStr1* is alphabetically less than *pszStr2*.	
Comments	Ignores case.	

SF_strlen

WORD SF_atoi (const char **pszStr*)

Returns the length of the specified string.

Parameters	*pszStr*	Pointer to the desired string.
Return Value	The length of string in bytes. In multi-byte systems, this may not necessarily equal to the number of characters in the string.	

SF_strstr

char * SF_strstr (const char **pszStr*, const char **pszToken*)

Searches for a sub-string within a string.

Parameters	*pszStr*	Pointer to the string to be searched.
	pszToken	Pointer to the sub-string to search for.
Return Value	Pointer to the first occurrence of *pszToken* in *pszString*; NULL if not found.	

SF_SysDateToInternalDate

DB_ITEM * SF_SysDateToInternalDate (DWORD *NumDays*)

Converts days since January 1, 1904, to the Satellite Forms internal date representation in YYYYMMDD format.

Parameter	*NumDays*	Number of days since January 1, 1904.
Return Value	Pointer to a string containing the Satellite Forms internal date representation.	
Comments	This function allocates memory with `SF_xmalloc`. Free this allocated memory with `SF_xfree`.	
See Also	SF_InternalDateToSysDate, SF_InternalTimeToSysTime, SF_GetSysTime, SF_SysTimeToInternalTime	

SF_SysTimeToInternalTime

DB_ITEM * SF_SysTimeToInternalTime (DWORD *NumSecs*)

Converts seconds since midnight to the Satellite Forms internal time representation in 24-hour HH:MM:SS format.

Parameter	*NumSecs*	Number of seconds since midnight.
Return Value	Pointer to a string containing the Satellite Forms internal time representation.	

Comments If the time passed to this function is more than one day's worth of seconds, the function keeps only the number of seconds beyond a whole number of days. As a result, this function accepts either seconds since midnight or the value returned by `SF_GetSysTime:` seconds since 00:00:00, January 1, 1904.

This function allocates memory with `SF_xmalloc`. Free this allocated memory with `SF_xfree` when you are done with it.

See Also SF_InternalDateToSysDate, SF_InternalTimeToSysTime, SF_GetSysTime, SF_SysDateToInternalDate

SF_TaskDelay

void SF_TaskDelay (DWORD *mSecs*)

Puts the handheld device in energy-conserving doze mode for the specified duration.

Parameter	*mSecs*	Duration of the delay in milliseconds.
Return Value	None	

SF_Tone

void SF_Tone (WORD *Frequency*, WORD *Duration*, WORD *Amplitude*)

Issues a tone of a specified frequency, duration, and amplitude.

Parameters	*Frequency*	Frequency of tone in Hertz.
	Duration	Duration of tone in milliseconds.
	Amplitude	Amplitude of tone (0 to 64).
Return Value	None	

Comments Pre-Palm III versions of handheld devices have all or nothing interpretations of the *Amplitude* parameter.

See Also SF_Beep

SF_UnlockRowItem(Obsolete. Do not use)

void SF_UnlockRowItem (void **pItem*, TABLE_REC **pRec*, WORD *RowNum*, WORD *ColIndex*)

Unlocks a data item that is locked with `SF_LockRowItem`. This is an internal system function and is not normally used.

Parameters	*pItem*	Pointer to a data item in database memory.
	pRec	Pointer to a table.
	RowNum	Zero-based row number of record containing the data.
	ColIndex	Index of column containing the data.
Return Value	None	

See Also SF_LockRowItem (Obsolete. Do not use), SF_RESIZELOCKEDRECORD (OBSOLETE. DO NOT USE.)

SF_UnqueryField

void SF_UnqueryField (TABLE_REC **pRec*, WORD *RowNum*)

Unlocks and releases a field accessed with `SF_QueryField`.

Parameters	*pRec*	Pointer to the desired table.
	RowNum	Row number of record to be unlocked and released.
Return Value	None	

Comments If you do not need to modify the data in the record, this function is a fast alternative to `SF_GetFieldCopy` or `SF_GetCachedField`.

See Also SF_QueryField

SF_ValidateField

CBOOL SF_ValidateField (CONTROL_REC **pControl*, TABLE_REC **pRec*)

Validates the contents of an Edit control to ensure the correct type.

Parameters	*pControl*	Pointer to the desired Edit control.
	pRec	Pointer to the desired table.

Return Value TRUE if the field contains data of the correct type; FALSE if it does not.

Comments This function fails, for example, if there is character data in a numeric field.

See Also SF_FormValidate

SF_xfree

void SF_xfree (void **pMem*)

Frees a block of memory allocated with `SF_xmalloc`.

Parameter	*pMem*	Pointer to a block of memory allocated with `SF_malloc`.

Return Value None

Comments `SF_xmalloc`, `SF_xrealloc`, and `SF_xfree` manipulate memory on the dynamic heap. Use these functions whenever you need to allocate small amounts of memory. Be aware that the dynamic heap is a very limited resource.

Memory allocated with these functions is writable when accessed through a pointer.

See Also SF_db_free, SF_db_malloc, SF_db_realloc, SF_xmalloc, SF_xrealloc

SF_xmalloc

void * SF_xmalloc (WORD *Size*)

Allocates a block of memory from the dynamic heap.

Parameter	*Size*	Requested size of the block of memory.

Return Value Pointer to the allocated block of memory if successful; NULL if the function fails.

Comments `SF_xmalloc`, `SF_xrealloc`, and `SF_xfree` manipulate memory on the dynamic heap. Use these functions whenever you need to allocate small amounts of memory. Be aware that the dynamic heap is a very limited resource.

Memory allocated by these functions is writable when accessed through a pointer.

See Also SF_db_free, SF_db_malloc, SF_db_realloc, SF_xfree, SF_xrealloc

SF_xrealloc

void * SF_xrealloc (void **pMem*, WORD *NewSize*)

Changes the size of a block of memory allocated with `SF_xmalloc`.

Parameters	*pMem*	Pointer to a block of memory allocated with `SF_xmalloc`.
	NewSize	New requested size of the block of memory.

Return Value Pointer to the reallocated block of memory block if successful; NULL if the function fails.

Comments The block of memory may move as a result of the reallocation.

`SF_xmalloc`, `SF_xrealloc`, and `SF_xfree` manipulate memory on the dynamic heap. Use these functions whenever you need to allocate small amounts of memory. Be aware that the dynamic heap is a very limited resource.

Memory allocated by these functions is writable when accessed through a pointer.

See Also SF_db_free, SF_db_malloc, SF_db_realloc, SF_xmalloc, SF_xrealloc

Chapter 13
Sample Application: Work Order

This chapter presents a relatively complex Satellite Forms application created with MobileApp Designer. Unlike Chapter , Quick Tour, which walked through the systematic construction of an application, this chapter focuses on demonstrating the capabilities of Satellite Forms. The vehicle for this demonstration is the Work Order application. The intention is not to have you duplicate the application, but to understand how it operates. This knowledge will be useful when you design and create your own applications for handheld devices with MobileApp Designer.

To illustrate the material that follows, open the Work Order project with MobileApp Designer, build it, download it to your handheld device, and then run the application as you read this chapter. This allows you see how the application works as you read about the construction and operation of its tables, forms, and controls.

If you installed Satellite Forms in the default installation directory, the Work.sfa project file is located in:
C:\Satellite Forms 8\Samples\Projects\Work Order\

The Work Order application supports operations at a fictional contracting company. The company's owner wanted an efficient way to accomplish the following tasks:

1. Assign individual client sites to crew bosses or individual workers.
2. Supply the crew boss or worker with easy access to contact and location data clients, as well as any important or useful information. For example, providing easily identifiable characteristics of the job site to help workers find it.
3. Provide the crew boss or worker with a list of tasks to be done for each client, complete with a summary description and important details about the job.
4. Give the crew boss or worker a place to record important information regarding clients or the job being done.
5. Provide the crew boss or worker with an easy way to note when a job is completed.
6. Furnish the company with a fast, efficient method of updating its work-order database and client information database after each day's work.

The Work Order application uses three tables and five forms.

The tables are:

- wrkSites: Stores the client contact and location information, the company ID number, and general notes regarding the client.

- wrkWorkItems: Holds information describing the work tasks, any special instructions for a task, and a value that signifies whether a job has been completed. The latter information combines with the wrkLookup table to determine the status to display for each task. The wrkWorkItems table also has a company ID column that identifies the client site associated with each job.
- wrkLookup: A lookup table that determines the status of a work task – To Do or Done.

The forms are:

- Main: The application's initial form. It contains the client Site List and buttons that access client information and the work tasks for each site. It also contains a hidden company ID number that is used to segregate the information displayed on the Site Summary and Work Item forms by individual client site.
- Info: Contains contact, company, address, and telephone information for each site listed on the Main form. It also contains a button that accesses the Notes form and a button that returns to the Main form.
- Notes: Provides a place for communicating or recording important or useful information about a site. It also contains a button that returns to the Info form.
- Site Summary: Contains a list of jobs for each client site. It also includes the job status: To Do or Done. Tapping any of the items in the list displays the Work Item form and with specific information for the task. The form also has a button that returns to the Main form.
- Work Item: Displays a summary of the task for a particular client site, check boxes indicating whether the job is complete, and more detailed information about the site and the task. For example, the work completed, where materials are located, what preferences the client has, and so on. It also has a button to return to the Site Summary form.

The remainder of this chapter presents the Work Order application in a manner that follows its most likely use, beginning with the Work Orders screen, which is the Main form in MobileApp Designer. Before discussing the forms, however, you need to look at how the tables are set up.

Work Order application: creating the tables

The first step in creating the Work Order application was to open and save a new project with MobileApp Designer. The next step was creating the three tables that contain the Work Order information and lookup values.

wrkSites Table

The wrkSites table, as shown in the following figure, stores the ID numbers (COMPANYID) for the client sites, contact and address information for the client sites, and a column (NOTES) for storing any outgoing data – instructions or directions transferred from the company database to the user's handheld device – or incoming data – notes about the client or the work the user entered and transferred back to the company database from the handheld device. The wrkSites table is the linked table for three forms: Main, Info, and Notes.

Figure 13.1 wrkSites Table Layout tab

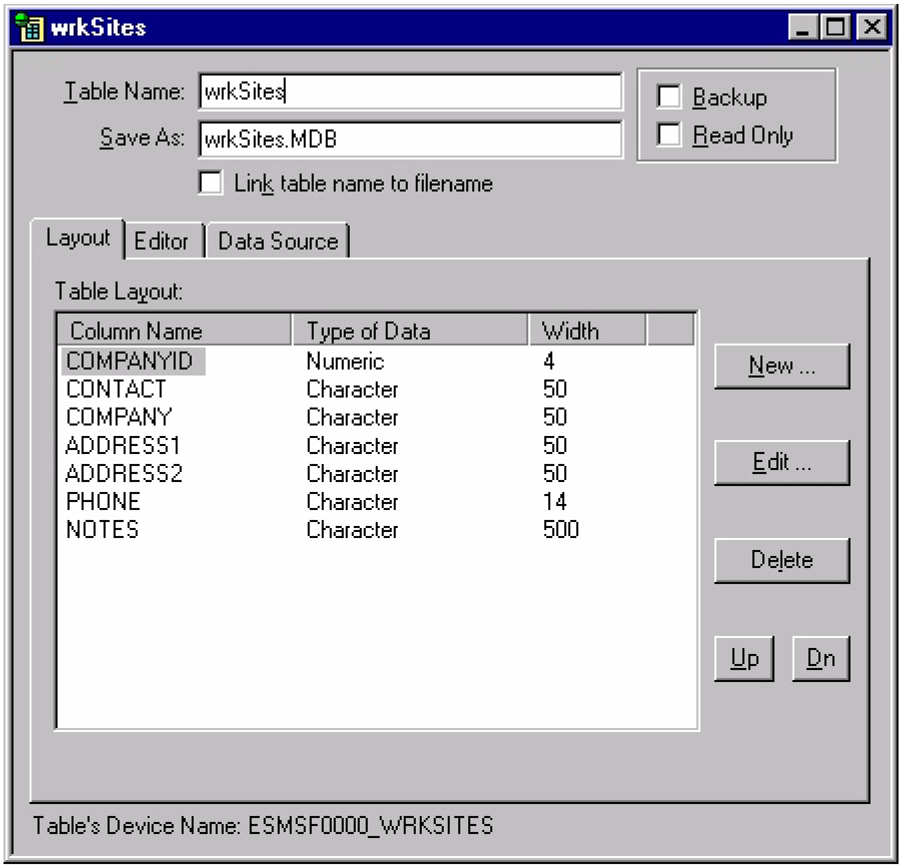


Table Name: wrkSites

Save As: wrkSites.MDB

□ Link table name to filename

□ Backup

□ Read Only

Table Layout:

Column Name	Type of Data	Width
COMPANYID	Numeric	4
CONTACT	Character	50
COMPANY	Character	50
ADDRESS1	Character	50
ADDRESS2	Character	50
PHONE	Character	14
NOTES	Character	500

Table's Device Name: ESMSF0000_WRKSITES

The sample data in the wrkSites table is shown in the following figure:

Figure 13.2 wrkSites Table Editor tab

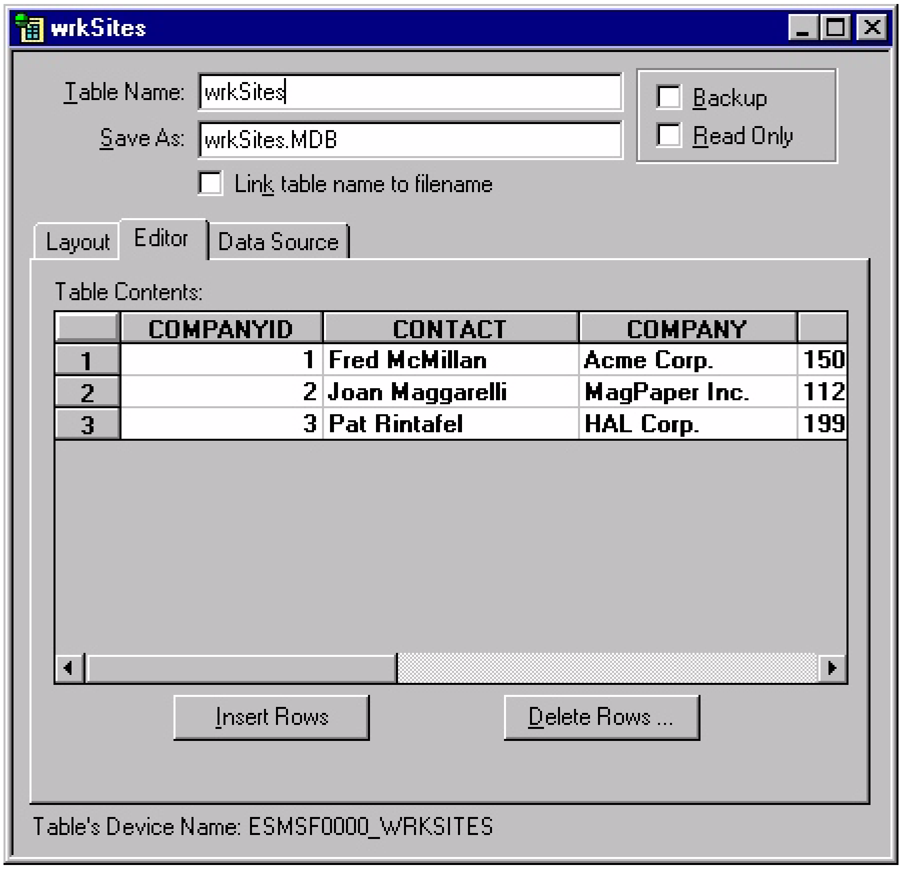


Table Name: wrkSites

Save As: wrkSites.MDB

□ Backup

□ Read Only

□ Link table name to filename

Table Contents:

	COMPANYID	CONTACT	COMPANY	
1	1	Fred McMillan	Acme Corp.	150
2	2	Joan Maggarelli	MagPaper Inc.	112
3	3	Pat Rintafel	HAL Corp.	199

Table's Device Name: ESMSF0000_WRKSITES

Since this data is used only to test the application on the handheld device, it only contains three records, one each for the companies Acme Corp., MagPaper Inc., and HAL Corp. In a real application, the information for the tables would come from a company database or from user input if the application were not meant to be integrated with a DBMS. The contact and notes information is intended for the user. The company ID information is essential for the operation of some of the forms and controls used in this application.

wrkWorkItems Table

The wrkWorkItems table, as shown in the following figure, is the linked table for the Site Summary and Work Items forms. It contains two informational fields, SUMMARY and EXTRAINFO, whose main purpose is to present specific information about work tasks. The COMPANYID field appears in this table, as it did in the wrkSite table. As in that table, its presence in wrkWorkItems is an important identifier used to separate work items by site. The COMPLETED column stores the values for the Yes and No Radio Button controls on the Work Items form. The list box control on the Site Summary form uses the values in these controls to determine the work status display for a task: To Do or Done.

Figure 13.3 wrkWorkItems Table Layout tab

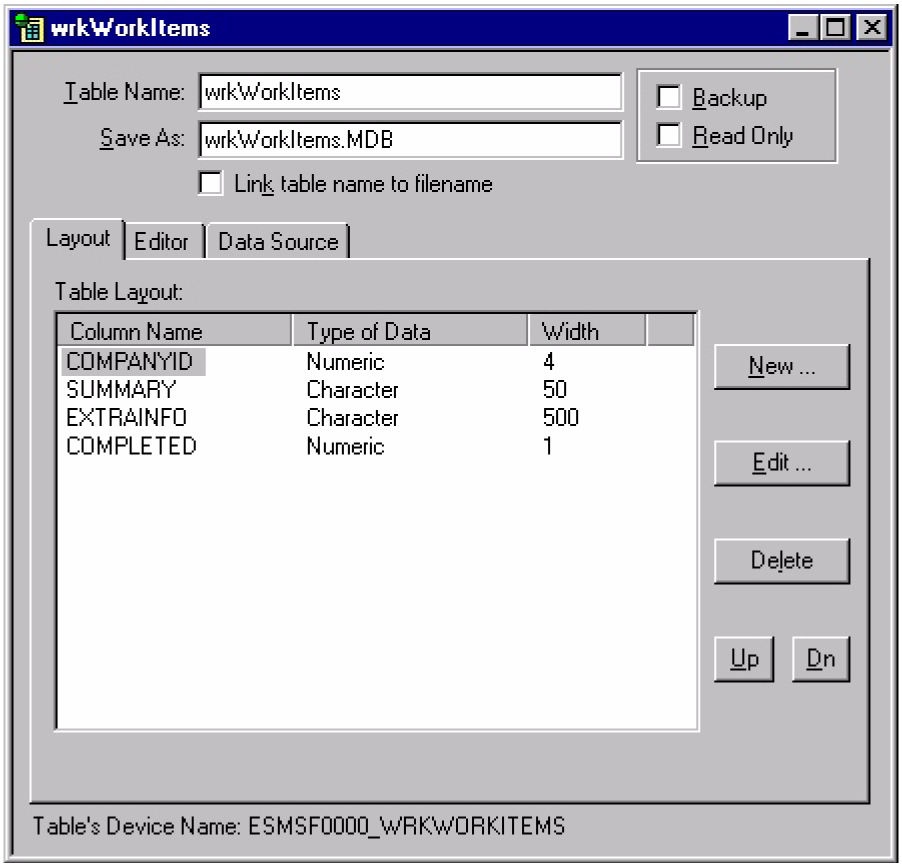


Table Name: wrkWorkItems

Save As: wrkWorkItems.MDB

□ Link table name to filename

□ Backup

□ Read Only

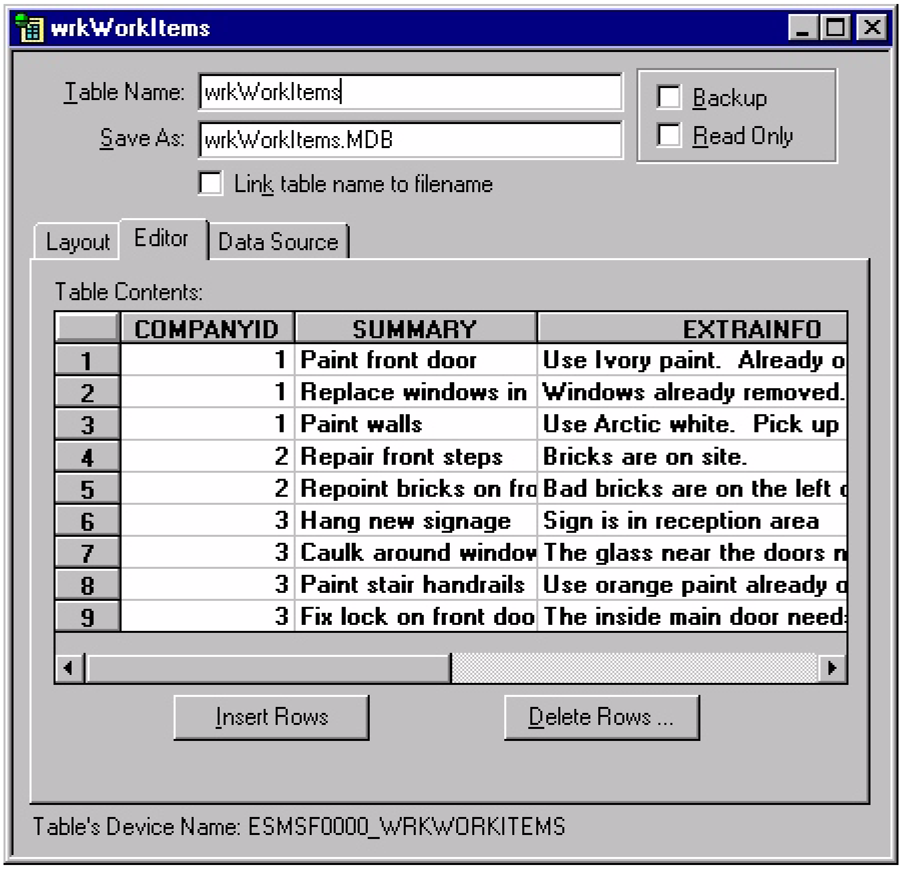
Table Layout:

Column Name	Type of Data	Width
COMPANYID	Numeric	4
SUMMARY	Character	50
EXTRAINFO	Character	500
COMPLETED	Numeric	1

Table's Device Name: ESMSF0000_WRKWORKITEMS

This table also contains some sample data suitable for testing purposes. As shown in the following figure, the SUMMARY column contains a brief description of work tasks, the EXTRAINFO column contains more detailed instructions for or information about the work, and the COMPLETED column stores the values of the Radio Button controls on the Work Item form.

Figure 13.4 wrkWorkItems Table Editor tab



wrkWorkItems

Table Name: wrkWorkItems

Save As: wrkWorkItems.MDB

□ Backup

□ Read Only

□ Link table name to filename

Table Contents:

	COMPANYID	SUMMARY	EXTRAINFO
1	1	Paint front door	Use Ivory paint. Already o
2	1	Replace windows in	Windows already removed.
3	1	Paint walls	Use Arctic white. Pick up
4	2	Repair front steps	Bricks are on site.
5	2	Repoint bricks on fro	Bad bricks are on the left c
6	3	Hang new signage	Sign is in reception area
7	3	Caulk around windov	The glass near the doors n
8	3	Paint stair handrails	Use orange paint already c
9	3	Fix lock on front doo	The inside main door need:

Table's Device Name: ESMSF0000_WRKWORKITEMS

In the sample data, the values in the COMPLETED column are all zeroes, which means no work items have been marked as completed. The COMPANYID number identifies the client site where the task needs to be done. The lookup control on the Site Summary form uses this number to look up the company name in the wrkSites table and display it on the handheld device.

wrkLookup Table

The third table is the wrkLookup table, as shown in the following figure.

Figure 13.5 wrkLookup Table Layout tab

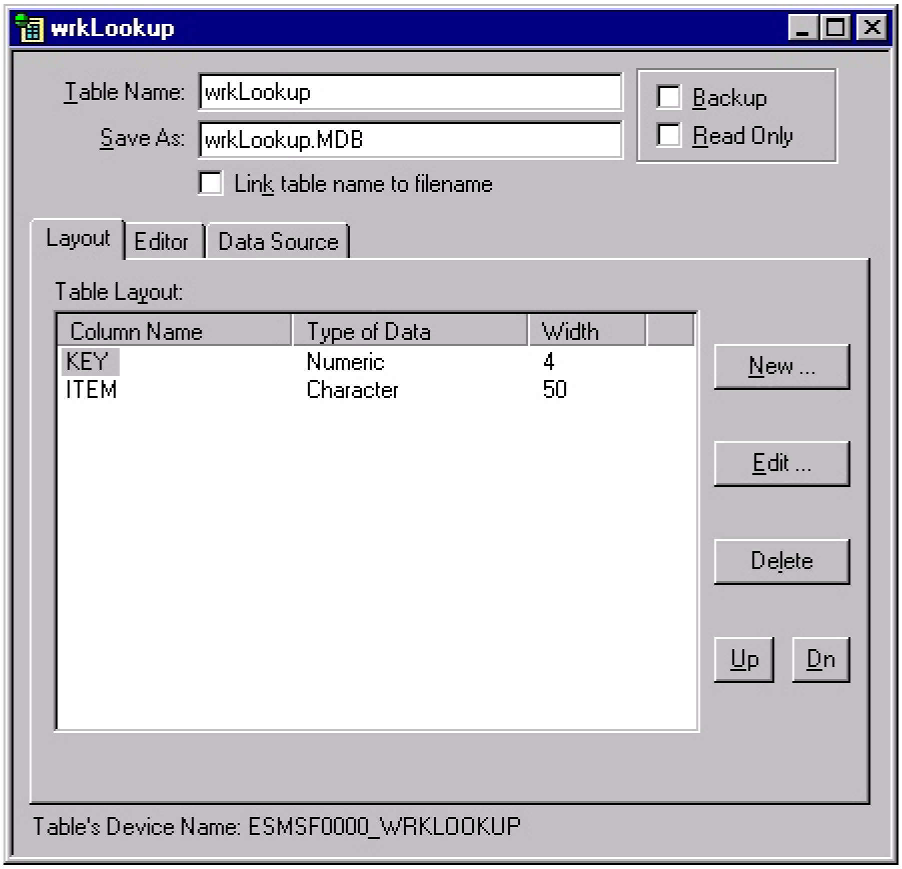


Table Name: wrkLookup

Save As: wrkLookup.MDB

□ Backup

□ Read Only

□ Link table name to filename

Table Layout:

Column Name	Type of Data	Width
KEY	Numeric	4
ITEM	Character	50

Table's Device Name: ESMSF0000_WRKLOOKUP

The wrkLookup table contains two columns – a numeric column labeled KEY and a character column labeled ITEM. As the title of this table indicates, this is a lookup table. The List Box control on the Site Summary form references it. This control takes the value stored in the COMPLETED column of the wrkWorkItems table, matches it to a value stored in the KEY column, and displays the ITEM text listed for that value on the handheld device's screen. The following figure shows the editor page for the wrkLookup table.

Figure 13.6 wrkLookup Table Editor tab

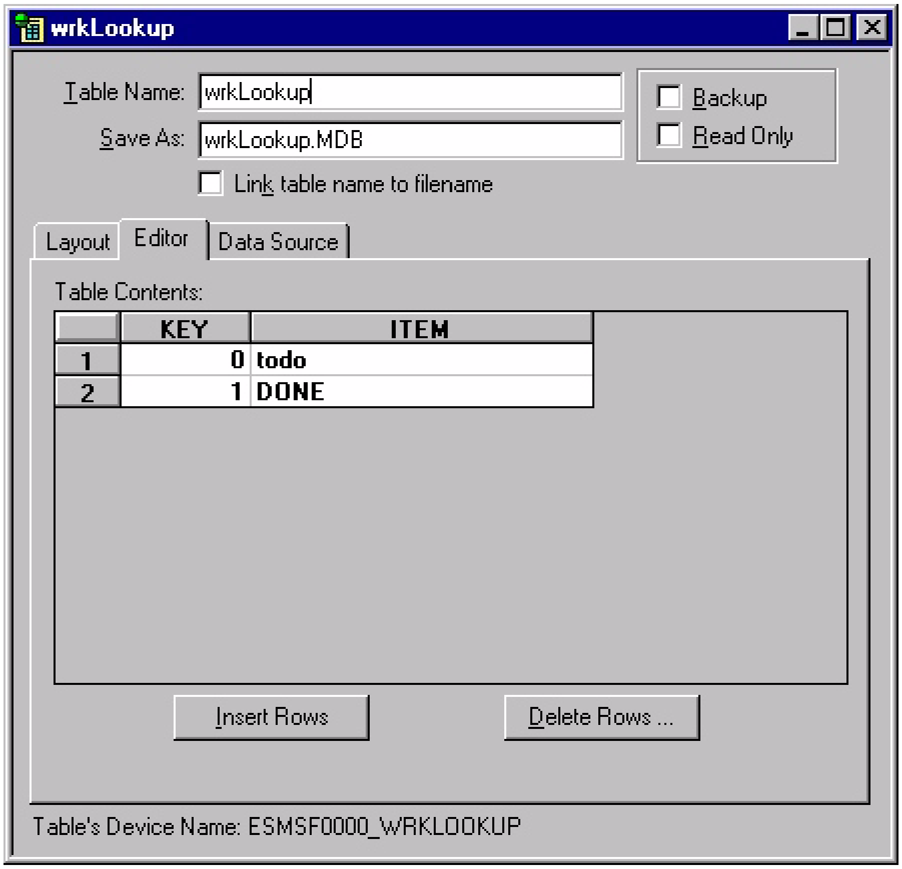


Table Name: wrkLookup

Save As: wrkLookup.MDB

□ Backup

□ Read Only

□ Link table name to filename

Table Contents:

	KEY	ITEM
1	0	todo
2	1	DONE

Table's Device Name: ESMSF0000_WRKLOOKUP

A KEY column value of zero corresponds to the ITEM To Do and a value of one corresponds to the ITEM Done. So if the COMPLETED column of the wrkWorkItems table contains a 0, the Site Summary form displays To Do as the work task status on the handheld device.

Work Order application: creating the forms

TheWork Order application uses five forms. The first form in the application is the Main form. This is the initial form the Work Order application displays on the handheld device. Its purpose is to show a list of client sites and give users options for viewing information about the site or information about the work list for that site.

The work list can be set up for each handheld user by downloading only the required information to a user's handheld device. For example, crew bosses would only have information about sites they are responsible for. For information on individualizing forms, see Chapter , Integrating with your Database, on page 195.

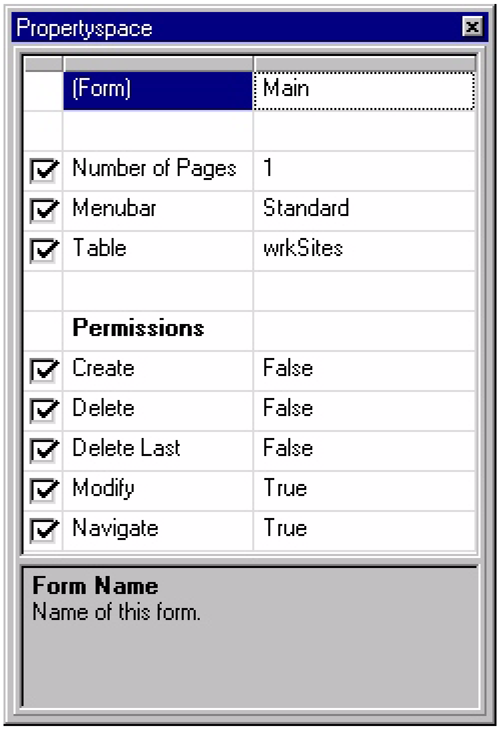
Creating the Main Form

The first step in creating the Main for is to open the default form, Form 1, and rename it **Main**.

Note Whenever you open a new project, MobileApp Designer creates a default form called Form 1. This form is also the default initial form for the application. When you rename the default form to **Main**, the initial form in the application properties automatically changes to Main. To use a form other than Form 1 as the application's initial form, select **Edit > Project Properties...** from the MobileApp Designer menu and select the form's name in the **Initial Form** combo box.

Next, set the Form Properties, as shown in the following figure:

Figure 13.7 Main form properties



Propertyspace

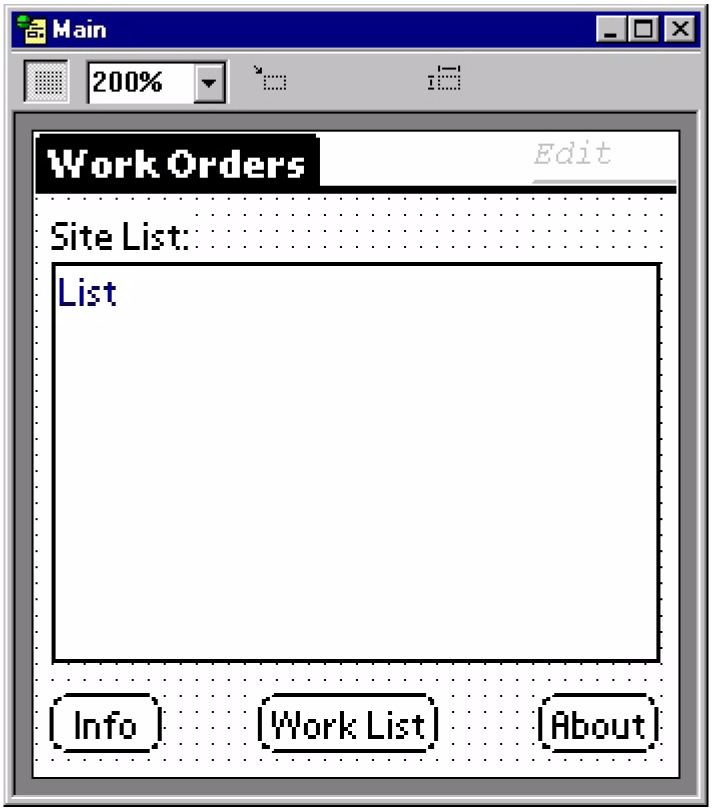
	(Form)	Main
☑	Number of Pages	1
☑	Menubar	Standard
☑	Table	wrkSites
	Permissions	
☑	Create	False
☑	Delete	False
☑	Delete Last	False
☑	Modify	True
☑	Navigate	True

Form Name
Name of this form.

The Main form is linked to the wrkSites table. Linking the form to a table activates the User Permissions, which are set as shown in the previous figure. The Create Record and Delete Record permissions are disabled to prevent users from adding or deleting clients or tasks. In the case of this sample application, it makes more sense to keep these functions in the hands of the DBMS manager. Users can change existing records, as indicated by the Modify permission, but they cannot add or delete them. For example, the crew boss or worker can mark whether a job is complete or add comments about the work or client, but cannot delete or add new clients or sites. This particular form has no controls that change the wrkSites table, so the **Modify** permission could be disabled. The Navigate permission allows users to scroll through records using the handheld device's scroll buttons. For this form, that means users can move through the Site List.

The Main form also contains several controls, as shown in the following figure:

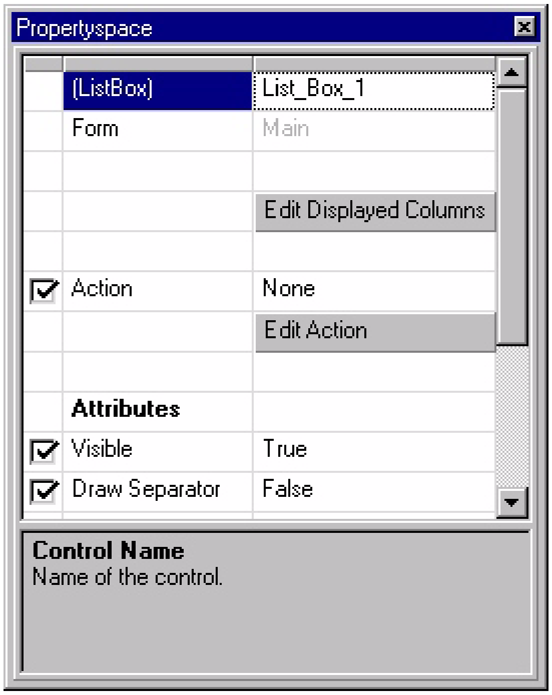
Figure 13.8 Main form controls



This form contains Title, Text, and List Box controls, a hidden Edit control, and three Button controls. The Title and Text controls are simply labels for the form and the Site List. The other controls perform various functions.

The List Box control displays the client sites. In the sample database, the three clients are Acme Corp., MagPaper Inc., and HAL Corp. The function of this control is to display these data items, which are drawn from the COMPANY column in the wrkSites table. The following figure shows the properties for the List Box control. It is set to display the contents of the COMPANY column and has no action when tapped.

Figure 13.9 Main form List Control properties



Propertyspace

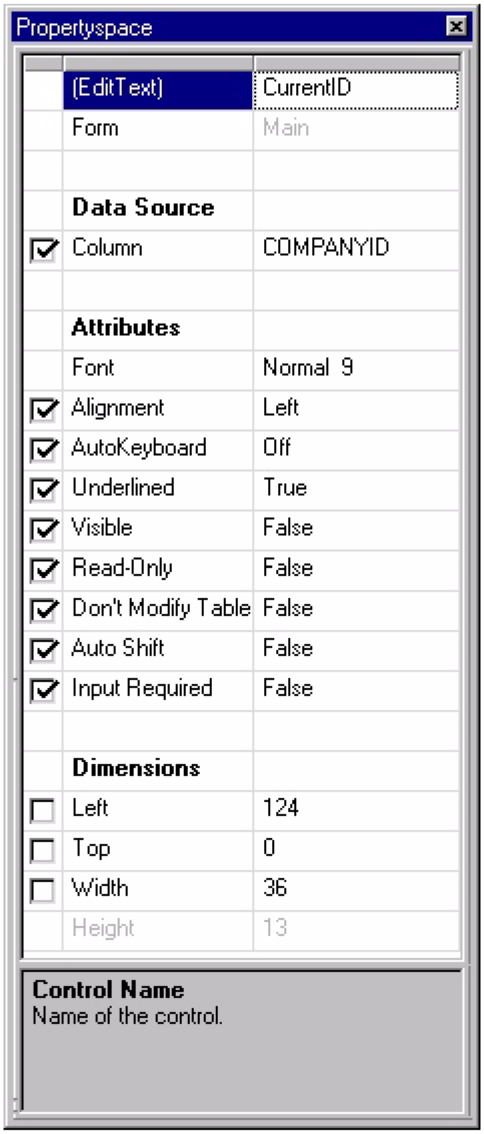
	(ListBox)	List_Box_1
	Form	Main
		Edit Displayed Columns
☑	Action	None
		Edit Action
	Attributes	
☑	Visible	True
☑	Draw Separator	False

Control Name
Name of the control.

The Main form also contains a hidden Edit control. It is visible in the upper-right corner of the Main form, as shown in Figure 13.8 on page 536. It does not, however, appear on the handheld’s screen.

The hidden Edit control is linked to the COMPANYID column in the wrkSites table and made invisible by setting the Visible property to False. The Propertyspace palette for the hidden Edit control is shown in the following figure:

Figure 13.10 Hidden Edit control properties



Propertyspace

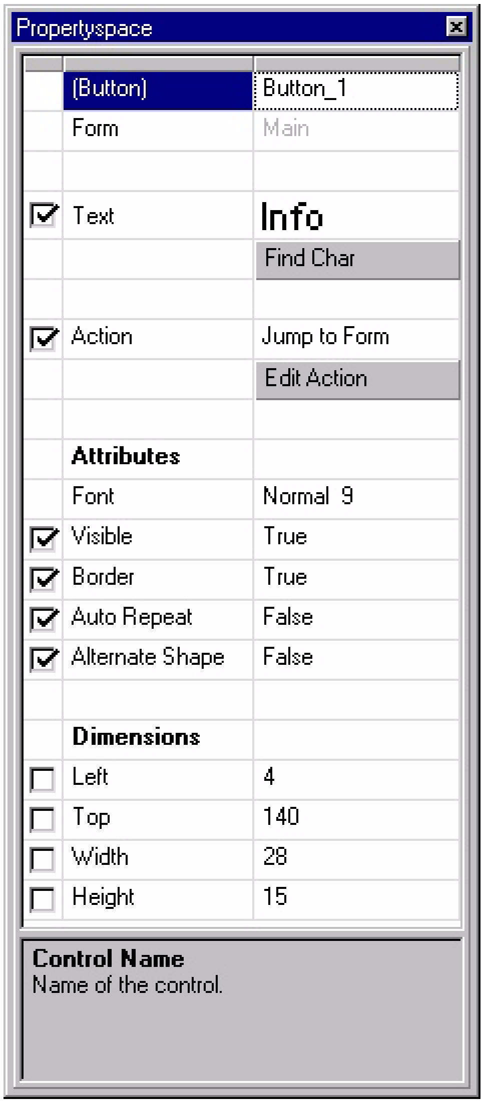
	(EditText)	CurrentID
	Form	Main
	Data Source	
☑	Column	COMPANYID
	Attributes	
	Font	Normal 9
☑	Alignment	Left
☑	AutoKeyboard	Off
☑	Underlined	True
☑	Visible	False
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Auto Shift	False
☑	Input Required	False
	Dimensions	
□	Left	124
□	Top	0
□	Width	36
	Height	13

Control Name
Name of the control.

The properties configuration shown above means the edit control contains, but does not display, the company ID of the form's current record. Using the database as an example, if the user selects Acme Corp. from the Site List, the CurrentID edit control contains a 1, since that is the COMPANYID value for that record in the wrkSites table. If the user selects MagPaper Inc. from the list, the CurrentID control contains a 2, and for HAL Corp., it contains a 3. Remember that selecting an item in a List Box control changes the current record.

Of the Main form's three button controls, the Info and Work List buttons are paired: one uses the CurrentID value for its operation and one does not. The About button displays version information about the Work Order application. The Info button is configured as shown in the following figure:

Figure 13.11 Info Button control properties



Propertyspace

	(Button)	Button_1
	Form	Main
☑	Text	Info
		Find Char
☑	Action	Jump to Form
		Edit Action
	Attributes	
	Font	Normal 9
☑	Visible	True
☑	Border	True
☑	Auto Repeat	False
☑	Alternate Shape	False
	Dimensions	
□	Left	4
□	Top	140
□	Width	28
□	Height	15

Control Name
Name of the control.

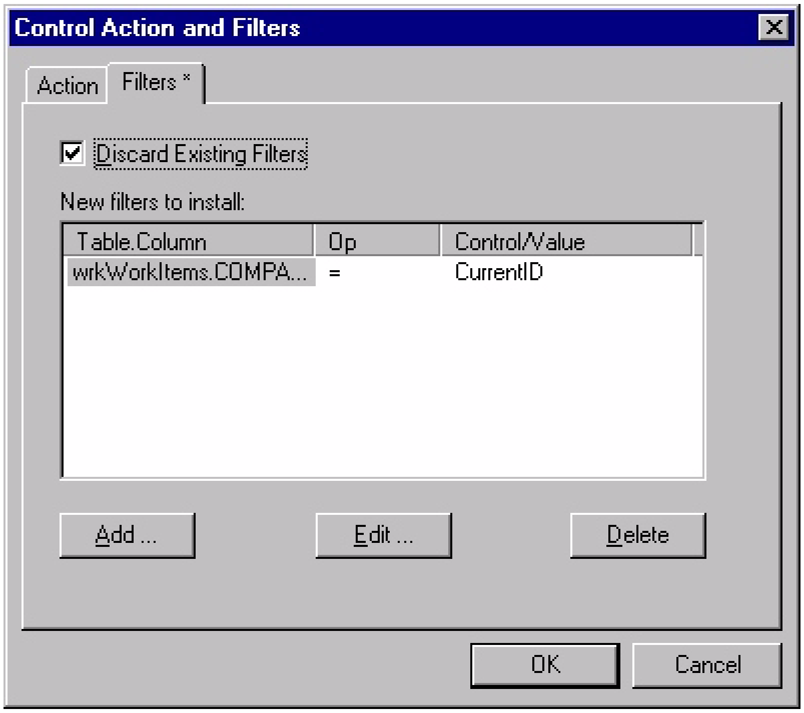
As shown above, this button's action is to jump from the Main form to the Info form, which shows the client contact and location information. The properties of the List Box control are set so that selecting an item from the list determines the form's current record. In other words, if the user selects MagPaper Inc. from the Site List, its record becomes the Main form's current record. When the user taps the Info button and jumps to the Info form, that form displays the contact and location information for the current record. Since this form and the Main form are associated with the wrkSites table, the current record for both forms is the same – MagPaper Inc.

The same is true when the user taps the Notes button to jump from the Info form to the Notes form. The Notes form is also linked to wrkSites, so its current record is the same as that for the Main and Info forms. Therefore, in this example, jumping to the Notes form displays the notes for MagPaper Inc.

The second button control on the Main form, labeled Work List, jumps to the Site Summary form. This button's properties are much like the Info button with two important exceptions: the Jump to Form action displays the Site Summary form and the Record Creation Option is Fail If No Records. The reason for the latter setting is that the user is not allowed to create a new record or jump to another form if no records exist.

The major difference between the Work List and Info Button controls is that Work List installs a filter, as shown in the following figure, after discarding any previously existing filters.

Figure 13.12 Work List table filter properties



☑ Discard Existing Filters

New filters to install:

Table.Column	Op	Control/Value
wrkWorkItems.COMPA...	=	CurrentID

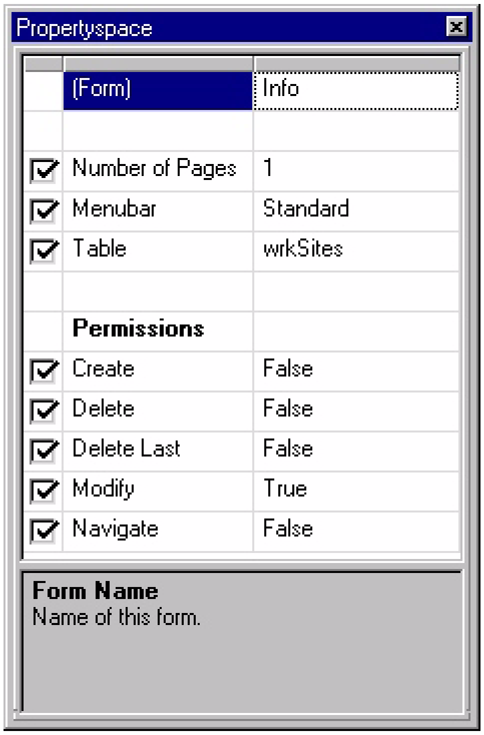
The filter is configured as follows. The table is wrkWorkItems, the column is COMPANYID, and the criterion is = (equals) snapshot of the CurrentID Edit control. As a result of these settings, tapping the Work List button clears any existing filters, installs a filter on table wrkWorkItems to show only records that match the value currently contained in the CurrentID Edit control, and then jumps to the Site Summary form. For the example above, MagPaper Inc. was the current record, so the CurrentID value would be 2. Therefore, when the jump to the Site Summary form takes place, the filter has already hidden all records in the wrkWorkItems table that do not have a value of 2 in the COMPANYID column. As a result, the Site Summary form only displays work tasks and status for jobs assigned to MagPaper Inc. In this application, this filter remains in place until it is replaced when the user returns to the Main form, selects a different company from the Site List, and taps the Work List button again.

Creating the Info Form

The purpose of the Info form is providing contact and address information for each client site. It also provides a button for accessing the Notes form, where the user can view or edit more detailed information about the client site.

The Info form's properties are set as shown in the following figure:

Figure 13.13 Info form properties



Propertyspace

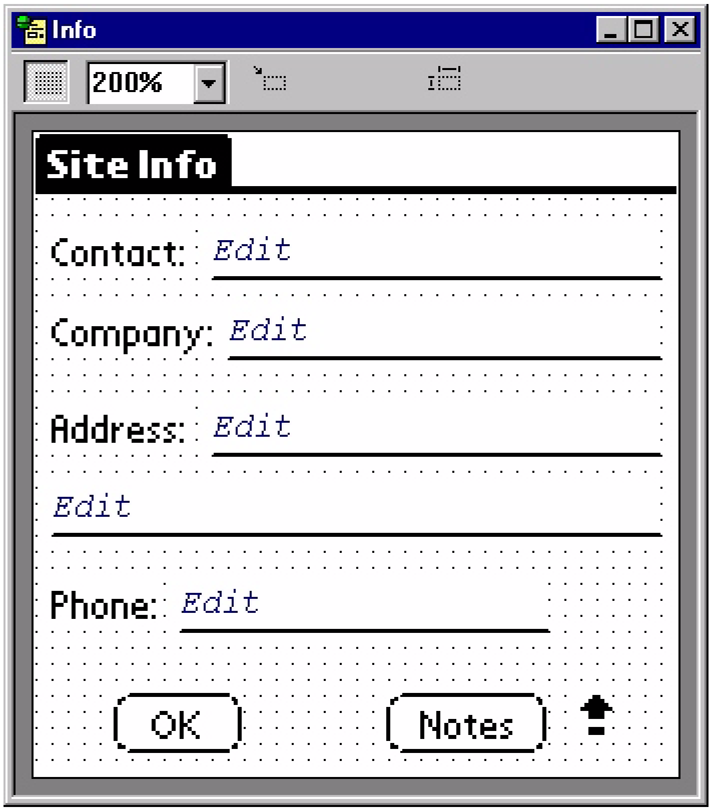
	(Form)	Info
☑	Number of Pages	1
☑	Menubar	Standard
☑	Table	wrkSites
	Permissions	
☑	Create	False
☑	Delete	False
☑	Delete Last	False
☑	Modify	True
☑	Navigate	False

Form Name
Name of this form.

The Info form is linked to the wrkSites table. The permissions for this form are set nearly the same as those for the Main form. The one difference is that the Navigate permission is disabled to prevent users from scrolling through records on this form. Since the user has already selected the desired company on the Main form, moving through the records on the Info form would cause the company information to be out of sync with the selected company. For example, the user could select MagPaper, Inc., jump to the Info form, access the next or previous record deliberately or by accident, and then jump to the Notes form, thinking the note was for MagPaper Inc. when in reality it might be Acme Corp. or HAL Corp. The client site name is not displayed on the Info form. Disabling the Navigate permission prevents this confusion from occurring.

The Info form contains thirteen controls, as shown in the following figure:

Figure 13.14 Info form with controls



The control configurations in this form are fairly simple. The four Text controls are labels for the Edit controls. The Edit control below Address does not need a label since it displays the second address line. The Edit controls are linked to columns in the wrkSites table that contain the data corresponding to the text labels. For example, the Edit control to the right of the Contact Text control is linked to the CONTACT column in the wrkSites table and displays that information for the form's current record.

The Graffiti Shift Indicator control in the lower-right corner of the form becomes active when the user enters data in the Edit or Paragraph controls and changes the shift mode of the handheld's Graffiti writing program. The indicator control displays different symbols depending on the shift state of Graffiti. For more information about Graffiti, see your handheld device user manuals.

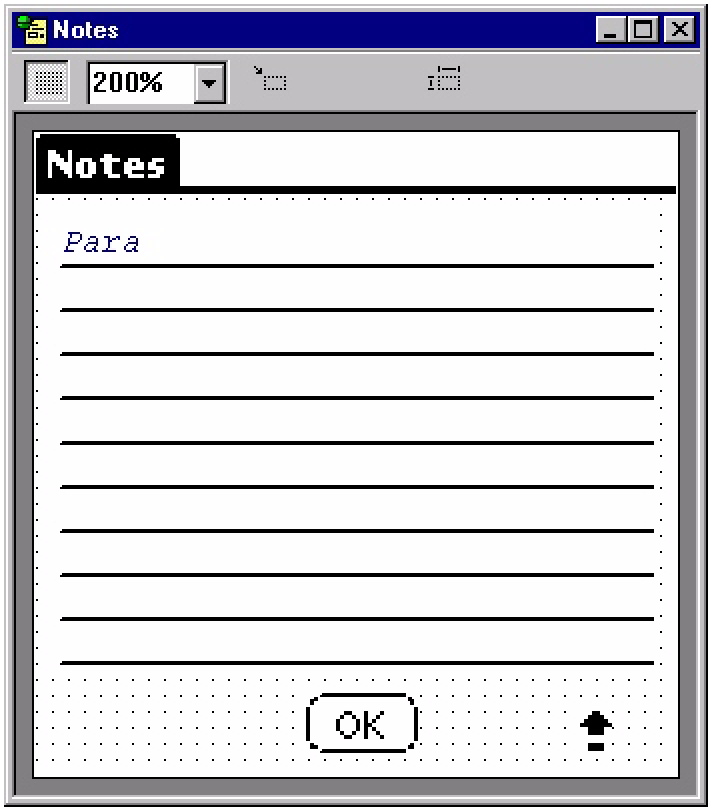
The OK Button control performs the action Return to Prev. Form, so tapping it on the handheld returns to the Main form. The Notes button control configuration is much like the Info button on the Main form, jumping to the Notes form. Since the Notes form is also linked to the wrkSites table, when the jump takes place the data contained in the NOTES column of the wrkSites table for the current record appears on the Notes form. If the current record for the Info form is MagPaper Inc., tapping the Notes button jumps to the Notes form and displays any notes entered for that company.

Creating the Notes Form

The Notes form provides the contracting company owner a way to convey important information about a site or client to the crew bosses and workers. It also gives the crew bosses and workers a way to record important data about a job or client that can be transferred back to the company database for review by the owner.

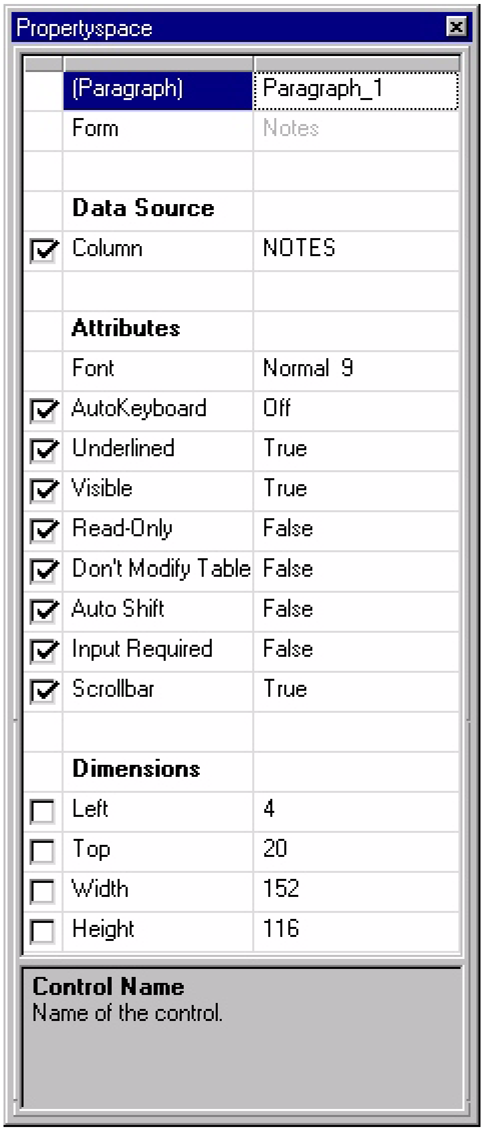
The form is simple, consisting of a Paragraph control for displaying or recording the notes, a Button control to return the user to the Info form, and a Graffiti Shift Indicator control, as shown in the following figure:

Figure 13.15 Notes form with controls



The Action property of the OK Button control is set to Return to Prev. Form, so tapping it displays the Info form. The NOTES column of the wrkSites table is the data source for the Paragraph control, so the Notes form displays the contents of that column for the current record. The Scrollbar property of the Paragraph control is also enabled, as shown in the following figure. Setting this property displays a scrollbar on the handheld device screen if the Paragraph control's contents are more than one page or screen can display.

Figure 13.16 Notes Form Paragraph control properties



	(Paragraph)	Paragraph_1
	Form	Notes
	Data Source	
☑	Column	NOTES
	Attributes	
	Font	Normal 9
☑	AutoKeyboard	Off
☑	Underlined	True
☑	Visible	True
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Auto Shift	False
☑	Input Required	False
☑	Scrollbar	True
	Dimensions	
□	Left	4
□	Top	20
□	Width	152
□	Height	116

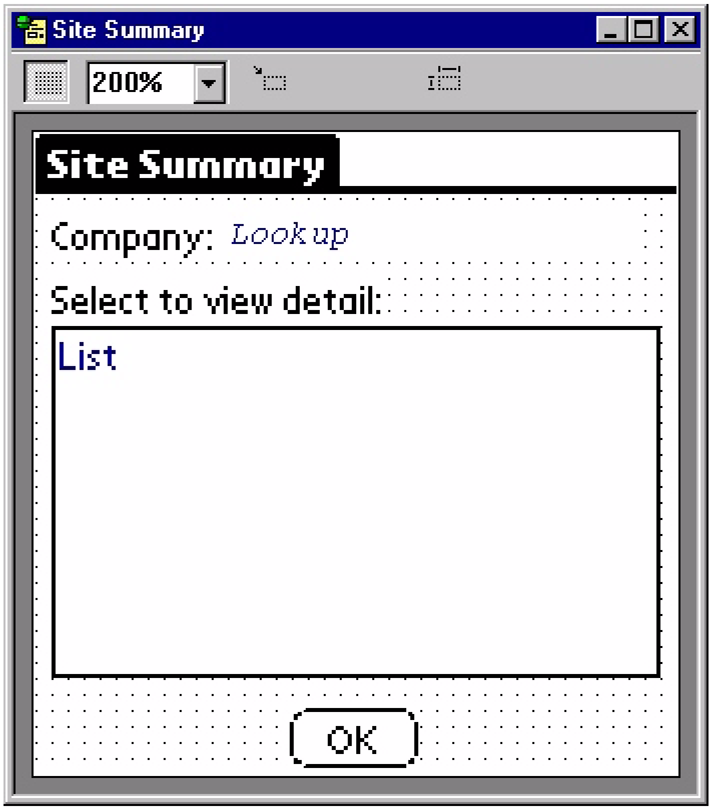
Control Name
Name of the control.

(The scrollbar feature only functions on PalmPilot.)

Creating the Site Summary Form

The next form is the Site Summary form, as shown in the following figure:

Figure 13.17 Site Summary form with controls

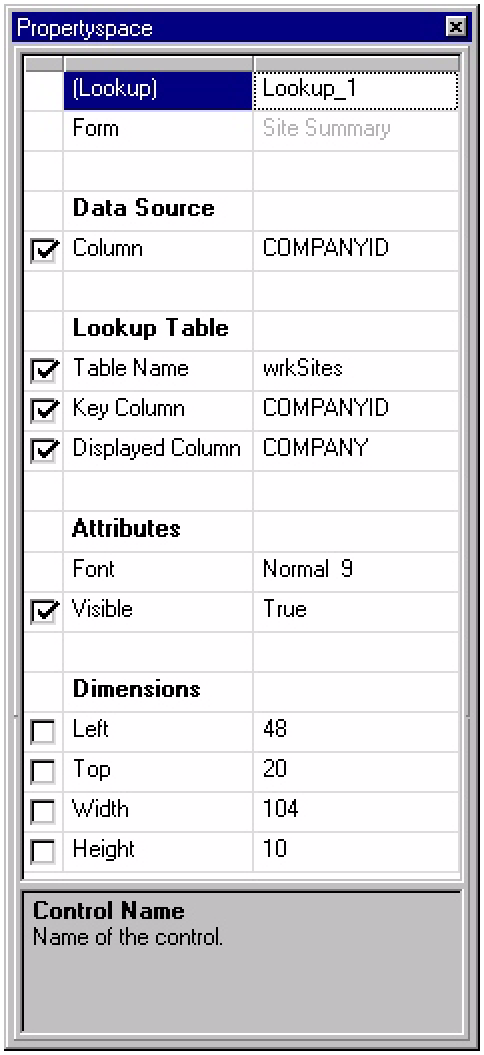


The purpose of the Site Summary form is to display a list of the work tasks for each client site along with the status for those jobs: To Do or Done. Tapping any item in the List control jumps to the Work Item form. This form displays a summary of the work that needs to be done, check boxes indicating the job status, and a place to view or record details about the job.

This form contains two Text controls for labels, a Lookup control, a List Box control, and a Button control configured to return to the Main form. The Lookup control and the List Box control require further explanation.

The properties for the Lookup control are shown in the following figure:

Figure 13.18 Site Summary Form Lookup control properties



	(Lookup)	Lookup_1
	Form	Site Summary
	Data Source	
☑	Column	COMPANYID
	Lookup Table	
☑	Table Name	wrkSites
☑	Key Column	COMPANYID
☑	Displayed Column	COMPANY
	Attributes	
	Font	Normal 9
☑	Visible	True
	Dimensions	
□	Left	48
□	Top	20
□	Width	104
□	Height	10

Control Name
Name of the control.

The data source for the Lookup control is the COMPANYID column in the wrkWorkItems table. The Lookup table properties are set as follows:

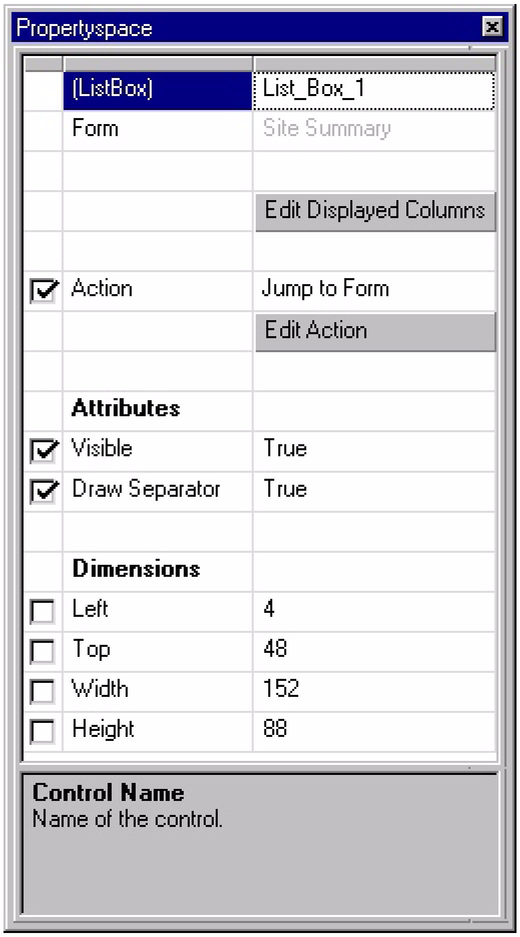
- wrkSites is the source Table Name – the control looks in this table for the data to display.
- The Key Column property is set to COMPANYID. This is the column shared by the wrkSites and WrkWorkItems tables. The Lookup control examines the value in the COMPANYID column of the wrkWorkItems table for the current record, compares it to the COMPANYID values in the wrkSites table, and chooses the first record that matches.
- The Displayed Column property is set to COMPANY. The Lookup control displays the contents of this column in the selected record using the wrkSites table on the handheld device.

Notice that the company name is not included in the linked table. The Lookup control fetches the company name for the client site selected on the Main form.

Tapping the Work List button on the Main form after selecting a client from the Site List installs the filter described under Creating the Main Form on page 534. The action of the filter limits the available data from the wrkWorkItems table to those records associated with the COMPANYID value in the Main form's CurrentID control when the user taps the Work List button. This means the Lookup control only finds one value in the COMPANY ID column of the wrkWorkItems table. In the earlier example, it would find 2, the COMPANYID value for MagPaper Inc. It would then reference the wrkSites table, find the COMPANYID value 2 in that table, and display the COMPANY column contents in that record – the company name MagPaper Inc.

The data displayed by the List Box control on the Site Summary form is drawn from the wrkWorkItems table, which is linked to the Site Summary form. As the following figure shows, the List Box control is configured to display two columns from the wrkWorkItems table – COMPLETED and SUMMARY. With the Work List button filter in place, these columns only contain the work summary and job status information for the client site selected from the Site List on the Main form. Returning to the example, this would be the work summaries and job status for MagPaper Inc.

Figure 13.19 Site Summary Form List Box control properties



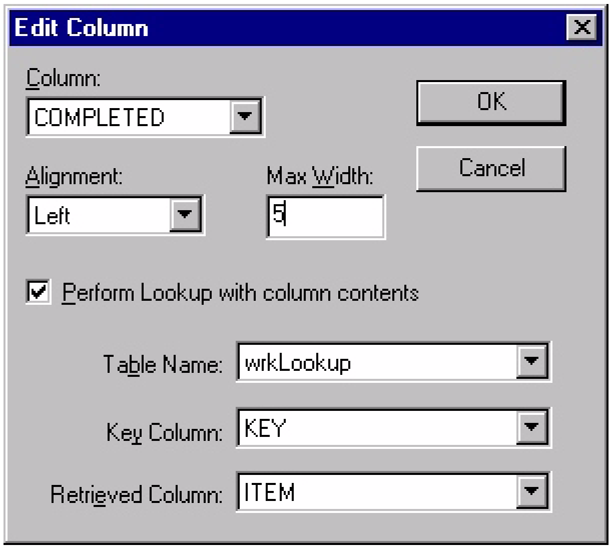
Propertyspace

	(ListBox)	List_Box_1
	Form	Site Summary
		Edit Displayed Columns
☑	Action	Jump to Form
		Edit Action
	Attributes	
☑	Visible	True
☑	Draw Separator	True
	Dimensions	
□	Left	4
□	Top	48
□	Width	152
□	Height	88

Control Name
Name of the control.

The SUMMARY column in the List Box control displays the SUMMARY column from the wrkWorkItems table, which is the Site Summary form's linked table. The COMPLETED column in the list box uses a table lookup function, as shown in the following figure:

Figure 13.20 Site Summary List Box control: COMPLETED Column Lookup properties



Edit Column

Column: COMPLETED

Alignment: Left

Max Width: 5

☑ Perform Lookup with column contents

Table Name: wrkLookup

Key Column: KEY

Retrieved Column: ITEM

Since one of the purposes of the Site Summary form is to display the To Do or Done status of a work task, the form needs a List Box control that checks and displays the status of a work task. When the user displays the Site Summary form, the List Box control references the value stored in the COMPLETED column of the wrkWorkItems table, compares that value against the values in the KEY column of the wrkLookup table, identifies the record in the wrkLookup table that matches that COMPLETED column value, and displays the ITEM column contents of the matching record in the wrkLookup table on the handheld device. The result is either To Do, if the value is 0, or Done, if the value is 1.

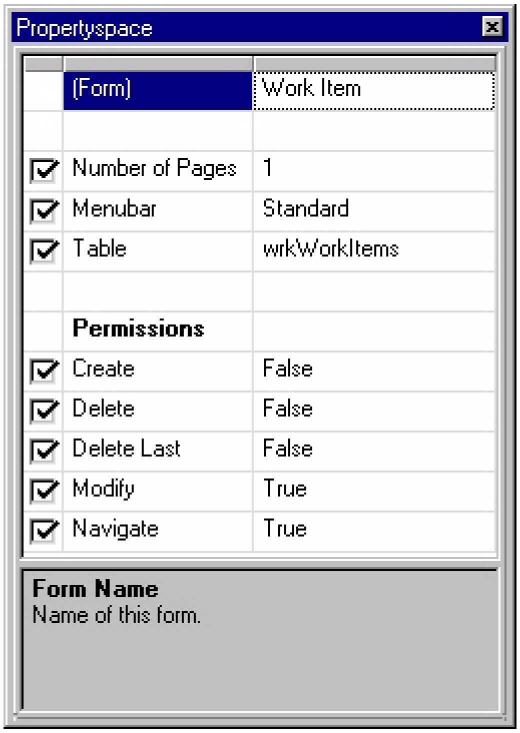
Finally, the List Box control action is Jump to Form with the Work Item form as the target. Tapping any item in the list jumps to that form, which then displays the summary, job status, and extra information for the selected work task. The Draw Separator Lines attribute separates the two columns when displayed on the handheld device.

Creating the Work Item Form

The last form in the Work Order application is the Work Item form. This form displays a summary of the work to be done for each task listed in the Site Summary form. It also provides a place to check off the job when it is complete and displays any detailed information or instructions relevant to the work task.

The Work Item form is linked to the wrkWorkItems table and configured as shown in the following figure:

Figure 13.21 Work Item Form properties



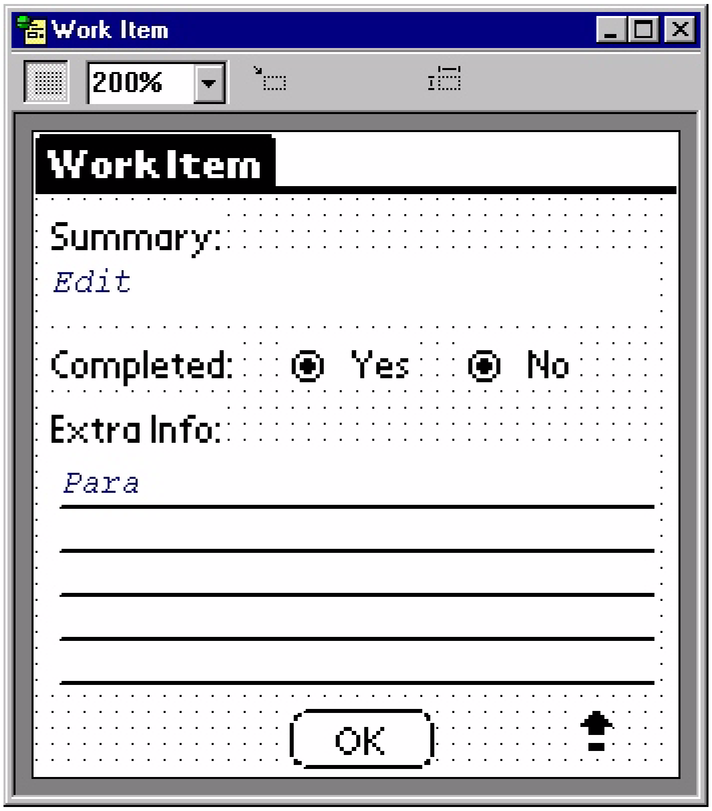
	(Form)	Work Item
☑	Number of Pages	1
☑	Menubar	Standard
☑	Table	wrkWorkItems
	Permissions	
☑	Create	False
☑	Delete	False
☑	Delete Last	False
☑	Modify	True
☑	Navigate	True

Form Name
Name of this form.

The Navigate permission is enabled in this form to allow users to scroll through the work tasks for a client site without having to return to the Site Summary form.

The Work Items form contains the controls shown in the following figure:

Figure 13.22 Work Item Form with controls



As with other forms in the Work Order application, the Work Item form uses Text controls to label the Edit controls. There is also a Button control configured with the action Return to Prev. Form, which in this case is the Site Summary form, and a Graffiti Shift Indicator control. There is also an Edit control that displays the work task summary. This control is linked to the SUMMARY column in the wrkWorkItems table.

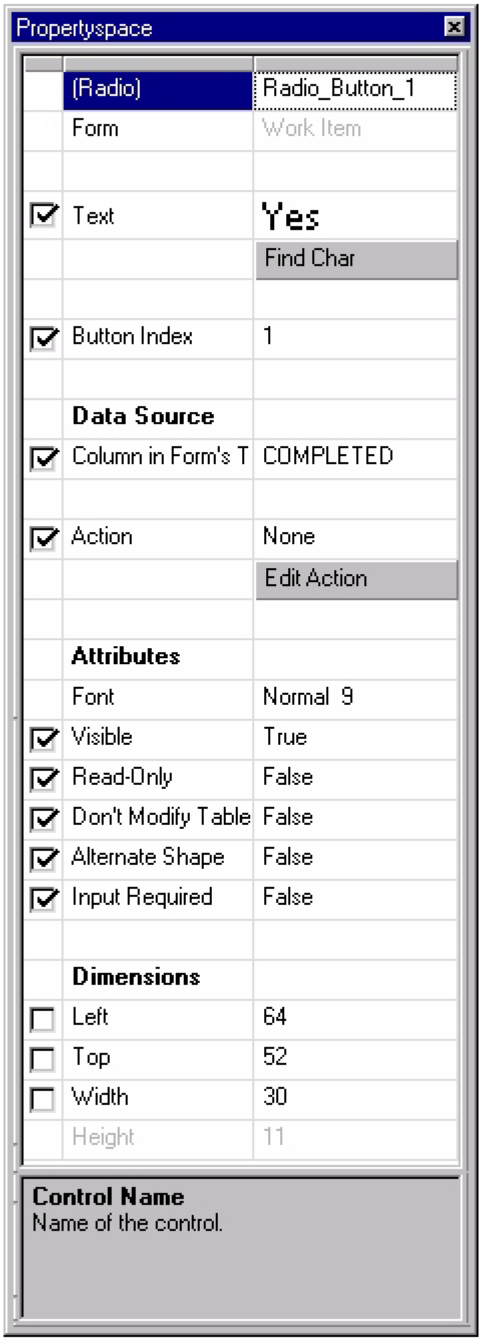
The Paragraph control is linked to the EXTRAINFO column of the wrkWorkItems table. This control displays any specific information or instructions related to a particular task. Finally, the two Radio Button controls are grouped, as described below, to provide a way of indicating the job status for a particular task.

First, it is important to understand how the Work Item form displays the information relevant to the job selected on the Site Summary form. The process begins on the Main form. When the user arrives at the Work Item form, the filter installed by the Work List button on the Main form is still in effect. Consequently, the records available to the form in the wrkWorkItems table are only those associated with the client site selected on the Main form.

Further refinement of the available information occurs in the list box control on the Site Summary form. As with the Main form, selecting a work task item from the **Select to view detail** list on the Site Summary form makes that item's record the current record. Since the Site Summary form and Work Item forms are both linked to the wrkWorkItems table, the current record remains the same when the user jumps to the Work Item form. When the user selects a particular job on the Site Summary form, for example, Repair front steps from the MagPaper Inc. list, the Work Item form displays the information relevant to that work task: job not completed, bricks are on site.

The purpose of the two radio buttons on the Work Item form is to record the job status for a particular task. This is accomplished on the handheld device by tapping the Yes or No item displayed on the Work Item screen. In the Work Order application, the Yes Radio Button control is configured as shown in the following figure:

Figure 13.23 Work Item Form Yes Radio Button control Properties



	(Radio)	Radio_Button_1
	Form	Work Item
☑	Text	Yes
		Find Char
☑	Button Index	1
	Data Source	
☑	Column in Form's T	COMPLETED
☑	Action	None
		Edit Action
	Attributes	
	Font	Normal 9
☑	Visible	True
☑	Read-Only	False
☑	Don't Modify Table	False
☑	Alternate Shape	False
☑	Input Required	False
	Dimensions	
☐	Left	64
☐	Top	52
☐	Width	30
	Height	11

Control Name
Name of the control.

The COMPLETED column of the wrkWorkItems table is the data source for the Yes Radio Button control and the Button Index property is set to 1. This tells the control to store a value of 1 in the COMPLETED column when the radio button is selected.

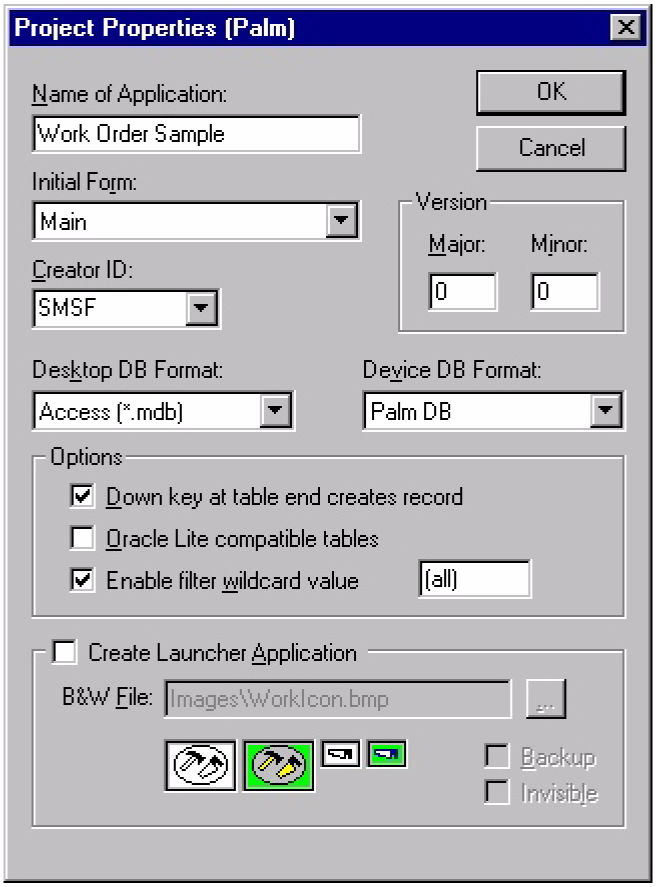
The No Radio Button control’s properties are identical except for the Button Index value, which is set to 0. When the user taps the No Radio Button control, it stores a 0 in the COMPLETED column. Since the COMPLETED column is the data source for both of these controls, they are grouped, which means only one can be selected at a time. The user can switch back and forth but cannot select both at the same time. When the user selects one of these radio buttons, it writes its value into the COMPLETED column, overwriting any previous values stored there. So, for example, when a crew boss completes a job and taps **Yes**, the Radio Button control stores a 1 in the COMPLETED column, overwriting the 0 the No radio button control previously stored.

The value stored by the action either Radio Button control on the Work Item form also determines the work status, To Do or Done, displayed on the Site Summary form for each work task.

Final steps

With the tables and forms complete, all that remains is to set the project properties. Select Edit > Project Properties... from the MobileApp Designer menu to display the Project Properties dialog box, as shown in the following figure:

Figure 13.24 Project Properties dialog box



Name of Application: Work Order Sample

Initial Form: Main

Creator ID: SMSF

Version — Major: 0 Minor: 0

Desktop DB Format: Access (*.mdb)

Device DB Format: Palm DB

Options
- ☑ Down key at table end creates record
- □ Oracle Lite compatible tables
- ☑ Enable filter wildcard value (all)

□ Create Launcher Application

B&W File: Images\WorkIcon.bmp

□ Backup
□ Invisible

The application name is Work Order Sample. This name appears on the handheld device in the Satellite Forms application list. Selecting the name from the list run the application.

The Main form is set as the application's Initial Form, meaning that the Main form is the first one displayed when the Work Order Sample application runs on the handheld device. Since the default Form 1 MobileApp Designer creates upon opening a new project became the Main form in the Work Order application, this item is already set. Using a different form as the application's initial form requires that you select the desired form from the Initial Form comebacks.

Conclusion

This example illustrated the following useful application design techniques:

- A List Box control configured with Jump to Form actions
- A List Box control associated with a lookup table to look up a value and display associated text
- A Button control configured to install a snapshot of control filter that limits form display contents
- A Lookup control configured to look up an ID number and display associated text

- Setting appropriate user permissions to improve the usability of your applications

Appendix A
Tips and Tricks

This appendix discusses how to perform some common operations with Satellite Forms. These operations include:

- Filtering information
- Creating unique record IDs
- Initializing new records
- Linking Drop List controls
- Creating active bitmaps
- Creating graphical Check Box and Radio Button controls
- Drawing on bitmaps
- Optimizing user permissions

Filtering information

Filters are useful in situations in which a table that contains many records but you only want to display items that meet certain criteria. See Using table filters on page 184 for more information on using filters.

As an example, open the **Restock** sample project using MobileApp Designer. **Select File > Open Project...** from the MobileApp Designer menu, navigate to the **Restock** sample project, and open the **Restock.sfa** file. If you installed Satellite Forms in the default installation directory, the Restock sample project file, **Restock.sfa**, is located in:
C:\Satellite Forms 8\Samples\Projects\Restock\

Tip Build and download the Restock application to your handheld device to see how the filter features work from the user’s perspective.

Open the **Main** form. When you select a customer using the **List_Box_1** control, the current row of the form changes to the row containing the selected customer name. Consequently, the hidden edit control called **CustID** in the upper right-hand corner always contains the contents of the CUSTID field for the selected customer.

When a user taps the **Orders** button, the application installs the following filter: `SrdOrders.CUSTID=CustID`. This means that in the SrdOrders table, all records in which the CUSTID field does not match the contents of the CustID edit control, at the time the user tapped the **Orders** button, are effectively removed from the view. While this filter is in effect, the SrdOrders table behaves as if it only contains records with the selected customer ID.

The Order button’s action property also jumps to the Orders form, which – because of the filter on its table – only displays the orders of the selected customer.

The **Done** button in the Orders form jumps back to the Main form and removes all existing filters, returning the application to its initial state.

Always remember to remove filters when they are no longer necessary or desirable. Leaving filters in place can cause unexpected behavior in other parts of an application.

Note that filters may also be managed from scripts, with the AddFilter, RemoveFilter, and RemoveAllFilters functions.

Creating unique record IDs

In database programming, you often need to insert a new record into a table. For example, if you need to create a new entry in a customer’s order, you must create an ID number for that item that does not repeat any of the IDs already in the table. This type of ID is called a unique ID.

You can use several methods to create unique IDs with Satellite Forms. Probably the most common method is to determine the largest numeric ID in use so far and add one to it. You only have to calculate an initial unique ID one time at application startup for each table that requires unique IDs and then simply increment the ID number every time you need a new one. The code below demonstrates this technique:

Put this code somewhere that runs one time at application startup:

```
'Calculate initial unique ID by getting
'the maximum value in the "ID" field.
UniqueId = Tables("Items").Max("ID")
```

Put this code where you create the record that needs an unique ID:

```
'Calculate a unique ID for the next new record.
UniqueId = UniqueId + 1
```

Initializing new records

Satellite Forms supports initializing the fields of new records with filters and scripts.

Initializing with filters

As an example, return to the Restock sample program referred to in Filtering information on page 555. Whenever a user is working with the Orders form, the `SrdOrders.CUSTID = CustID` filter is in effect. Tapping the **Add** button in the Orders form jumps to a new form, called Category, and creates a new record. When a record is created in a table, all filters currently applied to that table are scanned. If Satellite Forms finds a filter that applies to a table field and the filter has an equality (=) logical operation in its criterion, the field of the new record is initialized to the value or snapshot of the control specified in the filter criterion.

Notice, however, that since the filter applies to the table where in which the application is creating the record, the CUSTID column in the new record is initialized to the contents of the CustID control at the time the application installed the filter. In this case, this is the Customer ID of the customer currently being viewed. In this way, the Restock sample initializes the CUSTID column of new records to the current customer.

Initializing with scripts

Scripts provide two basic techniques for initializing records. The technique you use typically depends on whether a script or a control action created the record.

If you are creating records with script commands, simply assign the required values to the fields right after you create the record, as shown in the following example:

Example A.1 Script Example

```
'Create a record.
Tables("Tab1").CreateRecord

'Move table position to new record.
'Note: this does not affect the record displays on the form.
Tables("Tab1").MoveLast

'Generate next ID.
UniqueID = UniqueID + 1

'Initialize fields.
Tables("Tab1").Fields("ID") = UniqueID
Tables("Tab1").Fields("Urgent") = "F"
```

If you are creating records as part of a control action, you need to initialize the record in an event that occurs after the fact. For example, if you have a button on your form configured with the action Create Record, which creates a record in the table linked to current form, you could use code like this in your `AfterRecordCreate` event handler:

```
'Generate next ID.
UniqueID = UniqueID + 1

'Initialize fields.
Fields("ID") = UniqueID
Fields("Urgent") = "F"
```

You can apply the same technique to records created with a Jump to Form action. The `AfterRecordCreate` event fires in the target form after the action creates the record.

Linking Drop List controls

When you need to select from a large number of items, using a single drop list can be unmanageable. A better solution is to divide the items into categories and have two separate Drop List controls: one for the category and one for the items in that category.

Using this arrangement, you select the category from the category Drop List control, then the item from the item Drop List. Only the items in the selected category appear in the item list. Implementing this functionality requires filters.

As an example, assume we have three categories – animal, vegetable, and mineral – and six items – dog and cat (animals), tomato and potato (vegetables), and quartz and granite (minerals).

Set up two tables as shown in the following example:

Table A.1 Categories Table

CATID	CAT_NAME
0	Animal
1	Vegetable
2	Mineral

Table A.2 Items Table

CATID	ITEMID	ITEM_NAME
0	0	Dog
0	1	Cat
1	2	Tomato
1	3	Potato
2	4	Quartz
2	5	Granite

Add a Drop List control to a form, name it **CatDropList**, and set the following properties:

- **Table Name:** CategoriesTable
- **Key Column:** CATID
- **Displayed Column:** CAT_NAME

Add a second Drop List control to a form, name it **ItemsDropList**, and set the following properties:

- **Table Name:** ItemsTable
- **Key Column:** ITEMID
- **Displayed Column:** ITEM_NAME

To link the two Drop List controls, click the **CatDropList** control and click the **Edit Action** button. When the **Control Actions and Filters** dialog box appears, click the **Filters** tab. Click the **Add...** button, select **ItemsTable** from the **Table** combo box, select **CATID** from the **Column** combo box, select **equal (=)** from the **Show record if column contents...** combo box, and finally select **CatDropList** from the **...snapshot of control** combo box.

This sets a filter with the following criterion: `ItemTable.CATID=CatDropList`.

Now when you select a category, a filter is installed that hides all records in ItemsTable that do not have the selected category ID in the CATID column. Therefore, when you open the ItemsDropList control, you only see items in the proper category.

To complete this example, when you leave the form you should delete the filter you installed on ItemsTable. This is usually done as part of the action of the button that initiates a jump to the next form. You can remove this specific filter by installing a filter with the **del filter** criterion on ItemsTable.CATID, or remove all installed filters – assuming you no longer need any of them – by checking the **Discard Existing Filters** check box on the Filters tab of the Control Actions and Filters dialog box.

Creating graphical Check Box and Radio Button controls

Check Box and Radio Button controls support alternate shapes. The alternate shape for both controls is a box with text inside its boundaries.

To create a graphical Check Box or Radio Button control, set the Alternate Shape property to **True**, leave the text of the control blank, and place a bitmap inside the control's box on the form.

The Male and Female Radio Button controls in the Survey #2 program provide an example of this feature. If you installed Satellite Forms in the default installation directory, the Survey #2 sample project file, **TabSurvey.sfa**, is located in:
C:\Satellite Forms 8\Samples\Projects\Survey #2\

Drawing on bitmaps

Placing a bitmap on a form and then placing an ink control over it has a number of practical applications, for example:

1. You can place a thin dotted-line bitmap on a form to show users where to place their signatures.
2. You can create a form with a bitmap of a house and allow users to circle areas that are damaged or require maintenance.
3. You can create an accident report with a bitmap showing a street intersection. A police officer could then draw the cars' positions and insert arrows indicating events leading to the accident.

Note Only the user's pen strokes, captured in the Ink control, are transferred to the desktop computer. To show the Ink control on top of a bitmap on the desktop computer, the desktop application must explicitly overlay the two components.

Creating Color Icons for Palm with MS Paint

Procedure Create a color icon for a Palm OS Satellite Forms application.

1 Using MS Paint or any other bitmap editing application, open the desired template file.

If you installed Satellite Forms in the default installation directory, the Palm templates files are located in:
C:\Satellite Forms 8\Templates\

Note Do not resize the template file at any point in this process.

2 If desired, change the transparency color.

The transparency color determines the color that is invisible against the background. The default is bright green. Any pixels in the icon that you set to this color are transparent.

3 Use a drawing tool to draw the icon inside the icon bounding box in the template file. Only the graphics within the box are part of the icon.

Tip You can copy and paste an existing icon into the icon bounding box in the template file.

4 Use the color tools to set the colors in selected areas of the icon.

- To choose a different color, use the Pick Color tool to select the desired color from the palette.

5 When you are finished editing the icon image, save only the icon portion of the template file to a new name with a .BMP extension.

Refer to the instructions for your bitmap editing application for information on how to save only a portion of an image to a new file.

Optimizing user permissions

Setting a form's **User Permissions** properties correctly can greatly improve the usability of your applications. By default, most form permissions are enabled. When your application uses scripting or control actions, none of the form permissions apply.

Create Record/Delete Record

Set the **Create** and **Delete** record permissions to **False** when you have a list of items (records) that you want to preserve. Users can edit the existing records, but cannot add or delete records.

There are a number of reasons that this behavior may be necessary. For example, you have an application maintenance form designed specifically for adding and deleting records and you want users to access it whenever they want to add or delete records. You might have a warning message on that form that users should read before they add or delete, or some of your forms may not operate well when a record is deleted while the form is still in use. Another reason might be to prevent a novice user from deleting a piece of important information by mistake.

To see how disabling these permissions in selected forms can improve your application's usability, see the discussion under Navigate on page 561.

Delete Last Record

The **Delete Last** record permission is only relevant when a form has the **Delete** record permission enabled. Setting the **Delete Last** record property to **False** prevents the last record in a table, or the apparent last record if the table is filtered, from being deleted. Instead, when you attempt to delete the last record, the record is merely cleared. Consequently, the table linked to the form can never have its last record removed. More precisely, disabling this permission still allows the last record to be deleted but then immediately afterwards automatically creates a new record silently.

Disabling this permission is useful when it doesn't make sense for a particular form not to have any records in its linked table. Most forms need a record in which to save the contents of their controls. For this reason, the Delete Last record permission is disabled by default.

For an example, see the Customers application in the Samples directory. The Delete Last record permission is disabled in both forms because if the user deleted the last record, there would be no place to store any input entered into the forms. If this should be the case, the Satellite Forms Engine detects the illogical situation, displays a warning message, and discards any user input. You should design your application so that this kind of error never occurs.

If you installed Satellite Forms in the default installation directory, the Customers sample project file, **CustomersMDB.sfa**, is located in:
C:\Satellite Forms 8\Samples\Projects\Customers\Access 2000\

Modify

The **Modify** permission allows controls to change the form's underlying table. If you set this permission to **False**, any changes a user makes to a form are not saved. You can use this capability in a browsing form, where a user would be allowed to view information but not change it.

When you disable this permission, you should also consider disabling the **Create** record and **Delete** record permissions, make your Edit and Paragraph controls read-only, or both.

Navigate

The **Navigate** permission allows users to move through records on a form using the handheld device's scroll buttons. This permission is usually disabled to improve the usability of your application. If there are multiple forms linked to a particular table, your application will usually be more intuitive to users if you only give one of the forms the ability to navigate to the record being viewed. All other forms linked to the same table only display the record selected in the master form, that is, the with the **Navigate** permission set to **True**.

For an example, see the Customers application in the Samples directory. Only the Main form allows the user to create, delete, and navigate. If you installed Satellite Forms in the default installation directory, the Customers sample project file, **CustomersMDB.sfa**, is located in:
C:\Satellite Forms 8\Samples\Projects\Customers\Access 2000\

The Notes form, however, has none of these permissions enabled:

- Do not set the **Create** record permission to **True** because it is not intuitive to create a new customer record while viewing the Notes form. The user would see a confusing blank Notes form when what you probably want is to display a blank Main form to remind the user that a new customer record has been created.

- Do not set the **Delete** record permission to **True** for similar reasons. Deleting a record while viewing the Notes form simply show some other customer's notes after the current record is deleted. The user would have to go back to the Main form to find out which customer's notes are being displayed.
- Do not set the **Navigate** permission to **True** because the Notes form does not display the customer name. Paging through the notes without knowing which customer they belong to would be of limited value.

Glossary

Application: A complete system of tables and related forms, scripts, and extensions. The Satellite Forms MobileApp Designer creates custom applications that run on handheld devices.

Control: A user interface object placed on a form, such as an edit field or drop list. If the control can display data, the data comes from the current record of the form's linked table. The field associated with the control is the control's data source. A control that displays data from a data source is called a bound control, because it is bound to the data source.

Current Record: The record currently displayed on a form.

Database: A collection of related information stored for a specific purpose. In Satellite Forms, information is stored in a collection of database tables.

Device: A Palm Computing Platform or Pocket PC compatible handheld computer. (See handheld.)

DBMS: A database management system used to create database applications. Satellite Forms is a database management system.

Extension: Extensions come in two varieties: SFX plug-ins, and SFX Custom controls. Written in C by you or a third party, extensions can be used to manipulate data, perform complex business logic, and create custom controls and pop-up dialogs.

Field Type: An attribute assigned to a field that determines the type of data the field can contain. For example, character, numeric, date, and so on.

Field: A single element, or column, in a record. Each field has a unique name describing the type of value it contains. For example, in a sample CLIENTS table, the field CLIENTNAME contains the company name associated with each record. Note: The terms field and column are used interchangeably throughout this manual.

Form: A "page" or "screen" of information. In Satellite Forms, information can be contained in a form, or it can be retrieved from a table associated with the form. The table associated with a form is the form's linked table.

Handheld: A Palm Computing Platform or Pocket PC compatible handheld computer.

Record: A row in a data table containing a set of related fields. For example, , as shown in Figure 2.1 on page 21, an address or contact record typically contains name, company, address, and phone fields for a particular person or company.

Scripts: A set of Visual Basic-like language statements used to perform tasks. Scripts can add considerable functionality to your applications, including data validation, numeric calculations, and animations.

Table: An object that stores database information in one or more records (rows). Each record is separated into fields (columns).

Value: The information contained within an individual table cell. You can think of a value as the intersection of a record (row) and a field (column). For example, as shown in Figure 2.1 on page 21, Jack's Jokes is the value at the intersection of record 1 and the CLIENTNAME field.

Index