NS Basic/Symbian OS Handbook

January 15, 2009

©   NS BASIC Corporation, 2008.
71 Hill Crescent
Toronto, Canada M1M 1J3
(416) 264-5999

Every effort has been made to ensure that the information in this manual is accurate. NS BASIC Corporation is not responsible for printing or clerical errors. Specifications are subject to change without notice.

Canadian Cataloguing In Publication Data

Henne, George W.P., 1954-
   NS Basic/Symbian OS Handbook

ISBN 0-9695844-5-8

1. BASIC (Computer program language). 2. Windows (Computer programs).
I. Title

QA76.73.B3H46 2000   005.4'3 C00-930546-7

NS BASIC Corporation

http://www.nsbasic.com
support@nsbasic.com

LICENSE AGREEMENT

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE PRODUCT TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. License. The application, demonstration, system, and other software accompanying this License, whether on disk, in read only memory, or on any other media (the "Software"), the related documentation and fonts are licensed to you by NS BASIC Corporation ("NSBC"). You own the media on which the Software and fonts are recorded but NSBC and or NSBC's Licenser(s) and suppliers retain title to the Software, related documentation and fonts and are protected by United States copyright laws, by laws of other nations and international treaties. This License allows you to use the Software and fonts on a single Windows Product (which, for purposes of this License, shall mean a product bearing Microsoft's Windows logo), and make one copy of the Software and fonts in machine-readable form for backup purposes only. You must reproduce on such copy the NSBC copyright notice and any other proprietary legends that were on the original copy of the Software and fonts. You may also transfer all your license rights in the Software and fonts, the backup copy of the Software and fonts, the related documentation and a copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License.

2. Restrictions. The Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Software to a human- perceivable form. You may not modify, network, rent, lease, load, distribute or create derivative works based upon the Software in whole or in part. You may not electronically transmit the Software from one device to another or over a network.

3. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Software and related documentation and fonts. This License will terminate immediately without notice from NSBC if you fail to comply with any provision of this License. Upon termination you must destroy the Software, related documentation and fonts.

4. Export Law Assurances. You agree and certify that neither the Software nor any other technical data received from NSBC, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States. If the Software has been rightfully obtained by you outside of the United States, you agree that you will not reexport the Software nor any other technical data received from NSBC, nor the direct product thereof, except as permitted by the laws and regulations of the United States and the laws and regulations of the jurisdiction in which you obtained the Software.

5. Government End Users. If you are acquiring the Software and fonts on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees: (i) if the Software and fonts are supplied to the Department of Defense (DoD), the Software and fonts are classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software, its documentation and fonts as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and (ii) if the Software and fonts are supplied to any unit or agency of the United States Government other than DoD, the Governments' rights in the Software, its documentation and fonts will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA supplement to the FAR.

6. NS BASIC will replace at no charge defective disks or manuals within 90 days of the date of purchase. NS BASIC warranties that the programs will perform generally in compliance with the included documentation. NS BASIC does not warrant that the programs and manuals are free from all bugs, errors or omissions.

7. Disclaimer of Warranty on Software. You expressly acknowledge and agree that use of the Software and fonts is at your sole risk. The Software, related documentation and fonts are provided "AS IS" and without warranty of any kind and NSBC and NSBC's Licenser(s) (for the purposes of provisions 7 and 8, NSBC and NSBC's Licenser(s) shall be collectively referred to as "NSBC") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. NSBC DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR- FREE, OR THAT DEFECTS IN

THE SOFTWARE AND THE FONTS WILL BE CORRECTED. FURTHERMORE, NSBC DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE AND FONTS OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY NSBC OR A NSBC AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT NSBC OR AN NSBC AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. This limited warranty gives you specific rights. You may have others, which vary from state to state. LICENSOR'S ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT LICENSOR'S CHOICE, EITHER (A) RETURN OF THE PRICE PAID OR (B) REPLACEMENT OF THE SOFTWARE THAT DOES NOT MEET LICENSOR'S LIMITED WARRANTY AND WHICH IS RETURNED TO LICENSOR WITH A COPY OF YOUR RECEIPT. Any replacement Software will be warranted for the remainder of the original warranty period or 30 days, whichever is longer. This Limited Warranty is void if failure of the Software has resulted from modification, accident, abuse, or misapplication.

8. Limitation of Liability. Because software is inherently complex and may not be free from errors, you are advised to verify the work produced by the Program. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL NSBC BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF NSBC OR A NSBC AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. In no event shall NSBC's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Software and fonts.

9. Allocation of Risk: You acknowledge and agree that this Agreement allocates risk between you and NSBC as authorized by the Uniform Commercial Code and other applicable law and that the pricing of NSBC's products reflects this allocation of risk and the limitations of liability contained in this Agreement. If any remedy hereunder is determined to have failed of its essential purpose, all limitations of liability and exclusions of damages as set forth in this Agreement will remain in effect.

10. Support. NSBC may, at its option, provide support services at its standard fees for such services. Such support services will be governed by the limitations of liability under this Agreement.

11. Additional Restrictions: Any upgrade or enhancement of the program subsequently supplied by NSBC may only be used upon the destruction of the prior version, and shall be governed by the terms of this Agreement.

12. Controlling Law and Severability. This License shall be governed by and construed in accordance with the laws of the United States and the State of Delaware, as applied to agreements entered into and to be pered entirely within Delaware between Delaware residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

13. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Software, related documentation and fonts, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of NSBC.

TABLE OF CONTENTS

01. Introduction

NS Basic/Symbian OS is an integrated development environment (IDE) for developing computer programs that can be executed on the various models running Symbian OS. Throughout this manual, the various members of the family of products will be referred to as a "Devices".

NS Basic/Symbian OS allows the developer to develop programs on an Windows based personal computer. The IDE provides a visual programming environment where Device layouts are designed by dragging objects onto a simulated Device screen.

A BASIC programming language is provided to let the developer code program logic for actions to occur when the program starts, a form is loaded, an object is selected, etc. There are numerous commands and functions to control many of the Device features such as sound, screen display, and processing of Device resident files. There are a wide variety of mathematical and other built-in functions to assist the developer and minimize coding.

Completed programs can be downloaded to a Device for execution on the device.

01.A What is BASIC?

BASIC has been around for over 30 years. Over that period, hundreds of interpreters and compilers for BASIC have been developed, and a mountain of application code has been written. Many books continue to be published about the language. BASIC Special Interest Groups exist in a number of forms.

BASIC is somehow good for the soul. As new waves of languages come and go, BASIC still runs almost everywhere: without standards, it adapts to new environments easily and keeps pace with the fancy new languages. The ones that come and go.

Everyone, even Bill Gates, started with BASIC. Somehow, we all keep coming home to it over and over again. It's still the best language to get a program working quickly. It also has a gentle learning curve, making it easy to progress from a beginner to an advanced programmer.

The computer hardware that BASIC is programmed on has turned full circle since the days it was developed. The powerful language to which only the computer scientists and mainframe programmers had access can now be run on a hand held device.

01.B. What Devices are supported?

Applications written using NS Basic/Symbian OS will run on all devices using Symbian for UIQ 3 or S60 3rd Edition.

There will be slight differences in how some features work with different versions of the Symbian OS. In addition, some devices may need to have a Library installed to add support for features unique to that device.

01.C. What is StyleTap?

StyleTap is a runtime environment that provides services to NS Basic/Symbian OS. It provides a framework to allow program execution on multiple platforms. Originally based on the Palm OS API, it has a powerful and easy to use set of objects, plus support for communications, file handling and more. The StyleTap engine is installed with NS Basic/Symbian OS programs. It is tied to those programs: a full version is available from StyleTap, Inc. Programs written for NS Basic/Palm can in most cases be run in NS Basic/Symbian OS without changes. It runs behind the scenes and is not noticeable at runtime. More information is available at www.styletap.com.

01.D Installation and Support

Complete instructions for installation are in the ReadMe file that is included with the software. Please follow these instructions.

If you have problems during installation, check the following Tech Note for possible solutions:
http://nsbasic.com/symbian/info/technotes/TN01.htm

NS BASIC Corporation provides support for its products by email. Send your requests to support@nsbasic.com. We will do our best to get you a speedy answer.

NS BASIC also sponsors a very active Web Board. If you have questions on how to use NS Basic/Symbian OS, want Programming tips, techniques and info on NSBasic/Symbian OS, posting of sample code, announcements of cool apps developed with NS Basic/Symbian OS or have jobs or projects available or wanted, please visit the board:
http://groups.yahoo.com/group/nsbasic-symbian

If you are a new user or want to get the latest news on NS Basic/Symbian OS, choose "NS Basic Website" under the Help menu.

02. NS Basic Concepts

02.A Projects

NS Basic/Symbian OS saves your entire application (forms, program code, menu definitions, etc.) in a Windows desktop file called the project file (.prj). This file can be reloaded into the NS Basic/Symbian OS IDE for subsequent changes and re-generation of the downloadable Device application program files.

Project files are named

```
xxxx.prj
```

where "`xxxx`" is the project or application name as assigned by the developer.

02.A.1 Directory Structure

On the Windows development machine, NS Basic/Symbian OS retrieves and stores project files based on a global directory root. On installation, the default directory is

```
C:\NSBASIC
```

You may change the global directory path during installation. The global directory will have the following subdirectory structure:

\NSBASIC
`\Bitmaps` -used to store images (.bmp, .jpg, .gif)
`\Download` -used to store project output .prc, .sis and .sisx files
`\Lib` - used to save .inf files used by Libraries and Resources
`\Projects` -used to save project files (.prj and .cod)
`\Projects\Samples` -Sample code
`\Tools` - useful tools to help NS Basic/Symbian OS developers

You should always use the menu options in NS Basic/Symbian OS to change to a new global directory structure. NS Basic/Symbian OS creates the required subdirectory structure and uses your new directory as the default directory whenever NS Basic/Symbian OS begins execution.

02.A.2 UID3

Each program downloaded to a device requires a unique UID3 which is a 8 digit hex number. These numbers are allocated by Symbian.

If you are only creating programs for your own device, NS Basic/Symbian will take care of assigning a unique number to each new app. If you use the same code as another app, there will be interference between the programs.

If you are developing a program for distribution to other users, registration of the UID3 is a must. This minimizes the chances that your program will interfere with other programs and vice-versa. Check the chapter and Tech Note on "Signing your App" for more information.

The UID3 is entered as part of the project properties.

02.A.3 Launch Icons and Launcher Name

The Symbian Launcher is used to select an application to execute. It operates in two modes: Grid View, where each application is displayed on the Device screen with an icon and a descriptive name printed below the icon, and a List View where each application name and a smaller icon is drawn on one line of the screen. NS Basic/Symbian OS applications look like regular applications in the Symbian Launcher.

Icons for S60 and UIQ devices are formatted differently. Tech Note 9 explains how to create icons in the proper format for each device. Once you have created your icons, put the name of the icon file in the Launcher Icon S60 and Launcher Icon UIQ Project Properties.

If you do not supply an icon, NS Basic/Symbian OS will supply a generic icon.

You may also supply an optional application Launcher Name that will appear on the launcher screen under your icon. This can be specified in the Project Properties. If you do not supply a Launcher Name, the project Name is used automatically.

02.A.4 Project Properties

The project properties are:

(Name)	The name of the Project
[Project Path]	The pathname to the main project .prj file.
Company	The name of the company which developed the app.
Install Folder	The name of the folder to install the app into in the Symbian Launcher. If nothing is specified, the app will be installed into the "Installat." folder.
Launcher Icon S60	An .svg file, created as per Tech Note 9.
Launcher Icon UIQ	A .bmp filename, as per Tech Note 9.
Launcher Name	The name that appears in the Launcher. If left blank, the Name is used.
License	The name of the .txt file containing the license agreement for the app. The user must click to agree to this as part of the installation.
Theme	The default color theme for the project. This defines the colors of buttons, text and the background. See the Theme Editor, 04.J.
Type	A 4 letter type for the application. This should be set to 'appl' for normal applications. No other values are used at this time.
UID3	Up to three 8 digit unique numbers for the app, separated by commas. If you supply just one value, the others will be created sequentially.
Version	The user specified version number of the app. This should be in the format "1,0,0" for version 1 of the app. Note the use of the commas!

02.B Forms

Each project has one or more forms. A form is a collection of objects that appear on the screen at the same time. Each form must be given a unique name.

When you start a new project, a form named Form1 is automatically created. You may add additional forms to your project by using the Project Menu, right clicking on the Forms folder in the Project Explorer, or by pasting a form from another project.

A form can have code associated with it that runs before it is created, after it is created, or when it detects an event.

Use the NEXTFORM statement to move from form to form in your program.

02.B.1 Form Properties

The form properties you can edit are:

(Name)	The name of the form. Each form must be given a unique form name. Some commands require the use of the form name to identify the form to be acted upon. Forms are given a unique name by the IDE when a form is added to the project. You may change this name to be more descriptive.
[Id]	The unique id number assigned to this form by the IDE. This cannot be modified.
Default form	If True, this form will appear first when the app is executed
Height	Modal forms only. See Section 14 "Modal Forms".
Left	Modal forms only. See Section 14 "Modal Forms".
Modal Form	True/False. See Section 14 "Modal Forms".
Modal Tips	Text string with Tips message. See Section 14 "Modal Forms".
Nav Bottom ID	For platforms that cycle vertically, this property specifies which object receives the focus when navigating up from an object in the top row of the form (an object whose Nav Above ID is 0). A Nav Bottom ID value of zero means that focus does not cycle vertically in the form. See Section 15 "Navigation".
Nav First ID	ID of the object where focus is positioned when the form is initialized. If Nav First ID is 0, the operating system places the initial focus on the first action button, if there is one, or on the first object in the tab order if there is not. See Section 15 "Navigation".
Nav Flags	0: (Default) The Symbian OS will decide whether to start in object or application focus mode 1: The form will initially be in object focus mode. 2: The form will initially be in application focus mode. See Section 15 "Navigation".
Nav Jump To	ID of the object to which focus can "jump", if the device supports this feature. Devices can optionally have an action to trigger the movement of the focus to a commonly used object. See Section 15 "Navigation".
Show TitleBar	If True, the title is drawn on the top line of the form. If False, no title is shown and you can use the area for your own objects.
Title	The title to be printed across the top of the form (if Show TitleBar is True).
Top	Modal forms only. See Section 14 "Modal Forms".
Width	Modal forms only. See Section 14 "Modal Forms".

02.C Objects

Each form contains zero or more objects. These objects are used to display data and get input from the user.

To add an object to a form, select it from the Toolbox and click on the Design Screen to locate its top left corner.

Most objects can have code associated with them. The code is executed when the object is tapped or its value is changed by the user.

02.C.1 Object Properties

Object properties are edited in the IDE as you write the program. Most can only be set at development time: they cannot be modified at runtime :

(Name)	Every object is assigned a unique name. This name is used in commands to identify the object to be acted upon. An object is given a unique name by the IDE when the object is added to the form. You may find it useful to assign object names prefixed with an abbreviation of the object type. Suggestions are: but for button (Example: butQuit) fld for field psh for PushButton chk for checkbox lst for lists pop for popup sel for selector rpt for repeater scr for scrollbar lbl for label bmp for bitmap gad for gadget
[Id]	A unique identification number assigned by the IDE. This number is used internally at runtime to refer to the object. It cannot be modified.
Anchor Left	Controls how the object resizes itself when the label/text is changed. If True, then the left edge of the object is fixed; if False, the right edge is fixed.
Auto Shift	If True, the first character of words are automatically set to upper case at the start of an empty field, after a period or other sentence terminator or after two spaces.
Bitmap ID	The ID number of the bitmap that will initially be displayed in the object. The specified bitmap must be included in the project and be have the corresponding id number. When you include a bitmap file into the project, it is assigned an ID by NS Basic. The Project Explorer shows all bitmaps that have been included into the project and precedes the bitmap file name with the ID number assigned.
Cols	The number of columns of a Grid.
Dynamic Size	For Field objects with Single Line set to False. The field object's size expands to fit the size of the data that is entered as the field object's value.
Editable	For Field objects. If True, the object's value can be changed by the user. If not, the object's data is simply displayed.
Font ID	The font number to use in displaying the object's text label. Valid font numbers are 0 standard 1 bold 2 large 3 symbol 4 symbol 11 5 symbol 7 6 LED font 7 large bold font
Frame	Applies to button and repeater objects. If True, a rectangular frame is drawn around the object .
Group Id #	A number assigned to similar objects (checkboxes or PushButtons) when only one of

	the objects can be selected at a time. A Group Id # of zero indicates the object is not part of a group. A non-zero Group Id # is used to group objects so that only one is selected at a time. For example, if two PushButton objects are used to indicate a choice of YES or NO and only one of these PushButtons can be highlighted, they would both need to be assigned a matching Group Id #. Then, selecting one of the PushButtons would automatically deselect the other one. Values must be from 0 to 255.
Has Scrollbar	If True, a scrollbar object is attached to the field or grid object and the operation of the two controls is synchronized.
Height	The height of the object in pixels
Index	The index number of the object in the form.
Label	The text string displayed as part of the object
Left	The horizontal (x) coordinate of the left edge of the object's boundary (pixels)
Left Justified	If True, the field object's value is left-justified when displayed.
List	A list of possible values for the field.
Max Characters	The maximum number of characters that can be entered in a field object. The user is prevented from exceeding this number of characters..
Maximum	The maximum value of a scrollbar or slider.
Minimum	The minimum value of a scrollbar or slider.
Non-bold Frame	Applies to buttons and repeaters. If True, a 1-pixel rectangular frame is drawn around the object. If False, a 2-pixel frame is drawn. The Frame property must be True.
Numeric	If True, a field object can only have numeric values entered into it by the user.
Page Size	The increment added to or subtracted from the scrollbar or slider's value when its arrows are tapped.
Resource Type	A four character type identifier. Can be any 4 characters.
Selected	For Checkbox and Pushbutton. If set to True, the object is checked or highlighted.
Single Line	If True, the field object is limited to one single line of text. The user cannot enter Return or Tab characters. If False, the user may enter more than one line of data.
Top	The vertical (y) coordinate of the top edge of the object's boundary (pixels)
Type	The type of an object. Here are the possible values: 0 Field 2 List 4 Bitmap 8 Label 9 Form Title 10 Popup 11 Shift Indicator 12 Gadget 13 ScrollBar 20 button 21 choicebox 22 checkbox 23 popup trigger 24 selector 25 repeater
Underline	If True, the field object is underlined when displayed.
Visible	If True, the object is initially shown. Otherwise, it is hidden.
Visible Rows	The number of rows visible in a List or Grid object. If the number of items in the object exceeds this number, then scrollbars will be drawn to allow scrolling through the entire list. For practical purposes, the maximum value of this should be less than 30.
Width	The width of the object (pixels)
Nav Above ID	ID of the object that is above the current object. Should be 0 if the object is in the top row of the form. If the user navigates up from an object with 0 for its above object, some platforms will move the focus to the object specified by Nav Bottom ID in the form properties. See Section 15 "Navigation".
Nav Below ID	Nav Below ID: ID of the object that is below the current object. Should be 0 if the object is in the bottom row of the form. If the user navigates down from an object with 0 for its below object, some platforms will move the focus to the first object in

	the tab order. See Section 15 "Navigation".
Nav Flags	0: (Default) Nothing special to do with this field. 1: If this flag is set, the object is skipped when focus moves from object to object. The object is included in the order only when it explicitly gets focus (which primarily occurs when a field, table with a field, popup trigger, or selector trigger gets focus by being tapped with the pen). 2: Used with multi-line text fields, if this flag is set the field is put in interaction mode when it receives the focus. Otherwise, the field is drawn in the focused, non-interaction mode state when it receives the focus. See Section 15 "Navigation".

02.D Menus

A project may have one or more menus. The menus are arranged across the top of the screen with dropdown selections. When a selection is made, the code for the selection is executed.

Menus can be added from the Project menu , by opening the Menu Editor from the menu or by right clicking in the Explorer. Separators can be inserted to group the dropdown selections.

Within programs, menus are activated by capturing the menu key in a form's Event code and using the MENUDRAW command to display the menu. See Section 10.D for more information.

02.D.1 Properties

Menus, menubars and dropdowns have the following properties:

[ID]	The id of the item
[Proc Name]	The name of the code segment to run when clicked
Caption	The title on the menubar or dropdown selection
Name	The name of the menu
Shortcut	The shortcut key to the dropdown selection

02.E Program Code

Program code can be specified in a number of locations in an application program. The Project Explorer is an easy way to see which project components have code associated with them and to open up the Code Window to edit the code.

NS Basic/Symbian OS provides a programming language that is a BASIC-like syntax. There are a variety of language commands and built-in functions to ease the developer's task of writing code to control the program's execution. Program code can be supplied to either be invoked at certain points or on certain events that occur in the program's execution.

Program code can be associated with the following conditions or events:

1. code to be executed at the startup of the program. (Project Startup Code)
2. code to be executed at the end of the program. (Project Termination Code)
3. code to be executed prior to the display of a form. (Before Code)
4. code to be executed after all of a form 's objects are displayed (After Code)
5. code to be executed when events occur on a form: a key press, a tap, etc. (Event Code)
6. code to be executed when an object is selected. (Object Click Code)
7. code to be executed when a menu choice is selected. (Menu Dropdown Code)

The NS Basic/Symbian OS IDE provides an easy method for attaching program code where needed. An editor is provided to allow the developer to write and modify program code. NS Basic/Symbian OS contains an embedded compiler to parse program code and generate executable code for eventual download and execution on the Device.

Each section of code needs to start with a SUB statement that is matched with an END SUB statement. You may put more than one SUB or FUNCTION in a code module by starting the new code section after the END SUB of the existing code.

02.E.1 Project Startup Code

The Startup Code is executed when the application is launched.

To edit this code, select "Startup Code" from the Project menu, or right click on the project's name in the Project Explorer.

In this code, the developer may want to perform any program initialization such as opening of files, data variable initialization and declaring global variables. Since the forms and objects have not been created yet, they cannot be accessed.

02.E.2 Project Termination Code

The Termination Code is executed when the program is stopped. This can be caused when the program issues a STOP command or another application is launched.

To edit this code, select "Termination Code" from the Project menu, or right click on the project's name in the Project Explorer.

In this code, the developer may want to do any program termination tasks such as closing of files, serial ports and to save the current program state. See Section 10.G for more information.

02.E.3 Form Before Code

The Form Before code is executed as a form is opening, before it is displayed on the screen. It can be used to set up data and list objects. Avoid operations that affect a form's appearance in the Before code, such as objects or drawing. They will not work as the form has not yet been created.

The Before Code is executed automatically after a NEXTFORM command or after a REDRAW command.

It can be edited by right clicking on the form name in the Project Explorer or in the Design Screen.

02.E.4 Form After Code

The Form After code is executed after the form and its pre-defined objects are drawn. This code is useful to add more graphics or other form information to be overlaid on top of the pre-defined form layout and to fill in the values for listboxes, etc. Most functions are better done in the Form After code, since the object is then fully created.

The After Code is executed automatically after a NEXTFORM command or after a REDRAW command. It will also be called when the Symbian OS does a redraw. For example, some devices automatically do a redraw after a POPUPDATE or POPUPTIME function.

It can be edited by right clicking on the form name in the Project Explorer or in the Design Screen.

02.E.5 Form Event Code

The Form Event Code is executed when a form is the active form and

a. A key is entered or one of the special buttons/keys is pressed. Certain devices have jog dial switches and additional buttons that create events, or
b. The stylus or action button is pressed (PenDown) or lifted from the screen (PenUp), or
c. An event is sent to the program by a Library.

Form Event Code is executed before the standard Device operating system code for the same event. If you do not want the Device operating system to process the event, issue the SetEventHandled command in your event code which causes the operating system to ignore the event. The event code must be the first subroutine in the Form Event Code module.

02.E.6 Object Click Code

This code is executed whenever the object is selected by some action of the Device user (e.g. taps on a button, selects an item from a list, taps on a repeater symbol, etc.).

Object click code can be edited by right clicking on the object name in the Project Explorer, or by double clicking in the Project Explorer, or by double clicking on the object's outline in the Design Screen.

All object types except the shift indicator can have code that is executed when the object is selected.

In the case of a field object, the code is executed after the user positions on the field and exits the field to perform another operation (in other words, when a different object has been selected).

02.E.7 Menu DropDown Code

This code is executed on the Device when the user selects a given menu dropdown item.

The developer supplies this code by defining a menu, the menu bar elements, the dropdown choices for each menu bar element, and the program code for each dropdown choice. For more information on menus, see Section 10.D.

02.E.8 Code Modules

You may place common subroutines or other code in a separate code file (called a code module) and include that code module in one or more projects.

Code modules can be added to a project by choosing Add New Module or Add Existing Module from the Project Menu. They can also be added by right clicking on the Modules folder or Project Name in the Project Explorer.

Code modules do not become part of the project file; they remain as separate files. The project file contains a reference to the actual file name of the code module file. Code modules usually end in .cod.

02.F Bitmaps

Bitmaps (as well as .jpg and .gif files) may added to the project by using the Add Bitmap option in the Project menu or by right clicking on the Bitmaps folder or project name in the Project Explorer.

This will bring up a dialog box of the files in the Bitmaps folder. You may navigate to another folder to select images there. If you select multiple files, each will be loaded into the project.

NS Basic/Symbian OS does not physically include images until you compile the program to create the Device executable files, so the image can be modified at any time prior to compiling.

Usually, the images used will be 1 bit black and white images. Some devices allow you to use 2, 4, 8 and 16 bit color images. Using the Property Editor, you may add images of other bit depths after the initial image is added.

Each image is given a unique Resource ID as it is added to the project. Use this ID to identify it to a Bitmap object or a DrawBitmap command.

02.G Files

NS Basic/Symbian OS provides commands to create, read, write, and delete files on the device. The design of fields and content are determined by the NS Basic/Symbian OS developer. Files may be used to provide permanent storage of information on the Device. The files are in .pdb format.

Files can be created in sequential or sorted mode. A sorted files stays sorted as new records are added: there is no need to sort the file afterwards. For more information on Files, see Chapter 8. For historical reasons, files are also referred to as Databases.

02.H Resources

Resources may added to the project by using the Add Resource option in the Project menu or by right clicking on the Resources folder or project name in the Project Explorer. Resources may be Libraries, apps or files. This is a handy way to include these objects into a single executable for distribution.

This will bring up a dialog box of the files in the Lib folder. You may navigate to another folder to select resources there. If you select multiple files, each will be loaded into the project. NS Basic/Symbian OS does not physically include resources until you compile the program to create the executable files, so the resource can be modified at any time prior to compiling.

Each resource is given a Resource Type and a unique Resource ID as it is added to the project. You can use these in a DbCreateDatabaseFromResource statement, but it is usually not necessary. The resources you include will be installed as files in the same folder as your app.

03. The NS Basic/Symbian OS Language

03.A Variables

Variables are identifiers that can be assigned a value. For example, a variable named TODAYSDATE might be assigned a value of "10/15/98". A variable named PI might be assigned a value of 3.1415.

Valid variable names:
start with an alphabetic character
contain the letters a-z, the numbers 0-9, and the underscore character (_).
Have 30 or less characters

Variable names are case insensitive; the letters "A" and "a" will be equal in matching variable names. Therefore, the use of the variable name "COST" and "Cost" refer to the same variable. All variables must be defined using a DIM statement before they are used.

NS Basic/Symbian OS allows the following types of variables (see the DIM statement for further details):
Integer, Short - numeric values that are whole numbers without any fractional digits.
Date - values of year, month, and day stored in an internal format
Time - values of hour, minute, and second stored in an internal format.
Float, Double, Single - numeric values that may have fractional digits.
String, Byte – text values

Example

```
Dim Payment as Float
Payment = 2.00
If overdue then
  Payment=payment + .50
End if
```

In the above, payment is sometimes capitalized and sometimes not. All uses of the name 'payment' refer to the same variable.

A variable must be defined by a Dim or Global statement before it can be used in any command, arithmetic expression, or function. The order of compilation of code segments is:
1. the project level startup code
2. any separate code modules that are part of the project
3. the dropdown element code segments for all menu dropdown items
4. repeat for each form in the order forms are listed in the Project Explorer
5. the Before code for a form
6. the code for each object on the form

A good place for GLOBAL variable statements is in the project startup code. The form startup level is also a good place to put GLOBAL definitions for variables used only in that form and its object's code. Non-global variables in a code segment should be placed before the first use of the variable in other statements, so a good practice is to put all DIM statements at the start of a subroutine or function (immediately after the SUB or FUNCTION statement).

03.B Constants

The following kinds of constants are allowed

Integers

```
1
25
-1050
```

Floating point constants

```
1.0
32.5932
3.96
```

String Constants

```
"this is an example"
"San Francisco"
```

Hexadecimal constants (preceded by &h)

```
&h05
&he303
```

In most instances, a hexadecimal constant is treated as a string.

03.C Arrays

Arrays are variables that can have multiple values. A specific value is referred to by following the variable name with a subscript in parentheses. For example, the statement

```
DIM MonthlyCost(12) AS FLOAT
```

Defines a variable with 12 possible values. `MonthlyCost(1)` refers to the 1st value, `MonthlyCost(2)` the 2nd, and `MonthlyCost(12)` the 12th value.

Arrays start with subscript number 1 as the first value. Subscript 0 or a negative number is invalid.

NS Basic/Symbian OS allows 3 levels of subscripting for an array:

```
DIM WarehouseCost(5, 12, 10) AS FLOAT
```

Might be used to specify an array to contain values for 5 years worth of data, 12 months in each year and 10 different warehouses.

03.D User Defined Variable Type Structures

Users can define their own data structures with the TYPE and END TYPE statements. For example, the following code defines a user data type (PersonInfo)

```
Type PersonInfo
  Name as string
  Address1 as string
  Address2 as string
  City as string
  State as string
  PhoneNo as Double
  DateOfBirth as date
End Type
```

You can then define variables of type PersonInfo as follows:

```
Dim Mother as PersonInfo
Dim Father as PersonInfo
```

An individual element of a Type structure is referenced by preceding the element name with the type structure name as follows:

```
Father.PhoneNo=3198287766
Father.City="Galesburg"

Msgbox "Mother's phone is " + str(Mother.PhoneNo)
```

Additionally, type structures and arrays can be used together, but types cannot be nested. Type structures may have arrays defined at the type structure level:

```
Type BankAccount
  AcctNo as integer
  Balance as double
  PastDueFees as Double
End type

Dim Accounts(5) as BankAccount
```

In this case, there are 5 BankAccount structures and an individual element in one is referenced by `Accounts.Balance(3)=200.00`

Additionally, a structure can contain elements that are defined as arrays:

```
Type FacultyMember
  Name as string
  Degrees(5) as string
End type

Dim Faculty(200) as FacultyMember
```

An individual element would be referenced such as:

```
Faculty.Degrees(5, 1) 'the 5th faculty member's 1st degree
```

03.E Filenames

A Filename is used when an external file is opened and operated on (read, written, etc.). File names have the same naming requirements as variable names (including the need to define file names by the Dim statement before using the file name in a statement.) When NS Basic/Symbian OS files are transferred to the desktop, the extension ".pdb" is added to the name; the ',pdb" extension is not used in an NS Basic program.

03.F Statements

A statement is a unit of code that defines one executable command. Examples are:
- an arithmetic assignment statement
- a command to play a sound
- a command to display another form
- a call to another subroutine

Commands in NS Basic/Symbian OS are not case sensitive, therefore, as an example, a command typed in as

```
MsgBox "This is an error message"
```

Is equivalent to

```
msgbox "This is an error message".
```

Capitals inside quote marks are preserved. To continue a line, use an underbar character (“_”) as the last character on the line to be continued. Put an @ sign in the first column to break on that statement if Debug Mode is turned on in Compile/Download Options. See Tech Note 35 for complete information on the Debugger.

03.G Subroutines

A Subroutine is a group of statements that can be invoked by a `CALL` statement or an event that triggers the call of the subroutine

Subroutine names follow the same rules as the naming of variables.

Subroutines are defined by an initial `SUB` statement and terminated by `an END SUB` statement.

Subroutines may be defined with parameters whose values are passed to the subroutine when called by the calling statement.

```
CALL Sub1(2.0, 12, Amt)
...
SUB Sub1(Rate as float, Months as Integer, Pymt as float)
  Pymt = Rate * Months
END SUB
```

This code would calculate a new value for `Amt` and store that result in the calling module.

03.H User Functions

Functions are similar to subroutines except that functions always return a value and functions can occur in calling statements wherever a variable name would occur.

Functions are defined by an initial FUNCTION statement and terminated by an END FUNCTION statement.

Functions return a value of a certain type (FLOAT, STRING, INTEGER, DATE, TIME, etc.). The type of result to be returned is specified in the initial FUNCTION statement using the 'AS *type*" format.

```
FUNCTION DeliveryDate as Date 'the function returns a date value
```

Functions may be defined with parameters to be passed to the function by the calling statement (exactly like Subroutine parameters).

```
Amt=Func1(2.0, 12)
...
FUNCTION Func1(Rate as float, Months as Integer) as Float
  Func1 = Rate * Months
END FUNCTION
```

This code calculates a new value using the passed input parameters of `Rate` and number of `Months` and stores the result in the calling program's variable called `Amt`.

03.I Built-In Functions

NS Basic/Symbian OS has many useful functions that are supplied with the software and can be used by the developer.

```
X=Sqrt(y)
```

`Sqrt` is a supplied built-in function that will calculate the square root of a number and return the result.

03.J Comments

You may add comments to your source program by preceding the comments by a single quote mark. On a line of code, the remainder of the line following the single quote mark is considered as a comment. Comments are ignored by the compiler. Comments are a good way to document complicated statements so that you can remember their purpose or others looking at your code may be better able to interpret what your program accomplishes.

```
' This whole line is a comment
call mySubroutine param1 'the rest of this line is a comment
```

03.K Objects

03.K.1 Properties

Properties are parameters which describe an object. For example, one property of a checkbox is whether it is checked or not. A property of a popup object is the text of the currently selected item. All objects have properties for x and y coordinates of the top left corner of the object's position .

Properties of an object can be referenced by the format objectName.propertyName. A property is an expression, not a variable. For example, it cannot be used as a parameter whose value gets changed in a call (i.e. dbRead(db,key,fld.text) will not work.)

Example

```
listCities.noItems
popupStates.text
fldSalary.text
```

Properties can be referenced to obtain a value or can be given a new value. For example, the statement

```
msgbox popNames.text
```

gets the value of the currently selected member of the popup list and displays it with the MSGBOX statement. The statement

```
fldCountryOfBirth.Text="Canada"
```

sets the property `'text'` of the field object to have a new value of "`Canada`".

03.K.2 Methods

Methods are actions that can be carried out on an object. For example, a list object can have all of its elements removed by the Clear method. A popup object can have a new item added to its popup list of items by the Add method.

Methods of an object are referenced by the format

```
objectName.MethodName optionalParameters
```

Example

```
ListBookTitles.Clear
ListBookTitles.Add "Huck Finn",3
FldPublisherName.setFocus
PopupCollegeCourseNames.Remove 4
```

The properties and methods for each object are defined in the Reference section. NOTE: For bitmaps, the show and hide methods only work the first way.

03.K.3 Referencing Properties and Methods

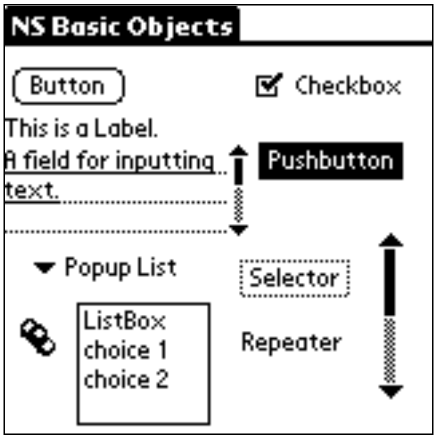
Besides the standard methods for referencing Properties and Methods, there are two other ways. The objectName in the above examples can be replaced by a string variable:

Example

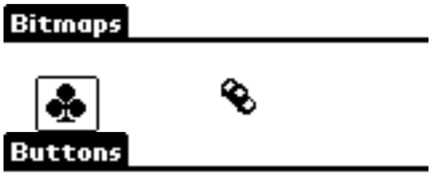
```
Dim obj as string
Obj="fldSalary"
Obj.text="New field value"
```

Properties and Methods can also be referenced from the Controls function. See "Methods" for more information.

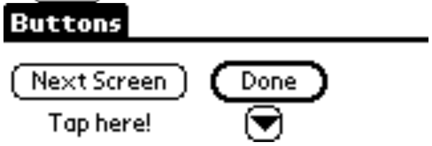
03.K.4 Object Descriptions



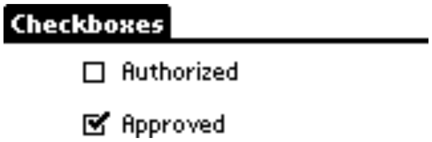
A form contains zero or more viewable objects each located at a Left and Top coordinate and possessing a height and width. NS Basic/Symbian OS provides a visual development environment where objects are placed on a layout by selection objects from a toolbox of available objects onto the Design Screen. This gives the developer a visual representation of how the objects will eventually appear when the app is executed on the Device.



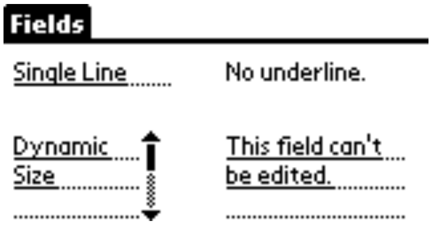
Bitmap - an area reserved on the form for a bitmap image to be displayed. It can display an image from a variety of different formats, or display text with gradients and coloring. Since it can respond to clicks, it can also be used as a button.



Button - A shaped object that has a text label displayed within its boundaries. A button is normally used to represent a command. If the button is tapped, the code for that button is executed.



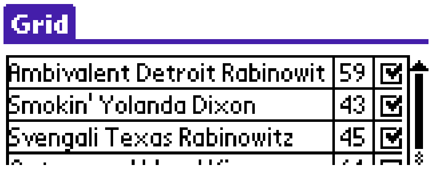
Checkbox - a small square rectangle followed by a text string which explains the purpose of the box. The checkbox may beset to True or False in which case a checkmark appears within the rectangle. If the checkbox is tapped, then the code that is defined for that checkbox is executed.



Field - An area where the user may enter text. Fields may be displayed with a default value or may be displayed as blank. Field objects may span one or more rows and can scroll.

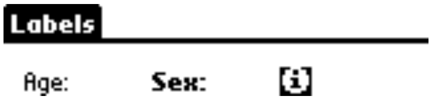
(not visible)

Gadget - a user defined object. If the gadget is tapped, then the code for that gadget is executed. A gadget object does not by itself have any visual properties and cannot be seen on the form. Any visualization within the gadget object's space on the screen must be accomplished by code that draws on the screen using the graphics commands. This is usually done in the form startup code. The other way for a gadget object to have visual properties is to have it occupy the same screen location as other objects (e.g. bitmaps or labels) that have visual properties.

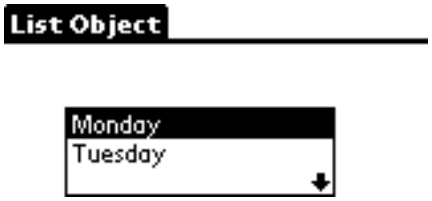


Ambivalent Detroit Rabinowit	59	☑
Smokin' Yolanda Dixon	43	☑
Svengali Texas Rabinowitz	45	☑

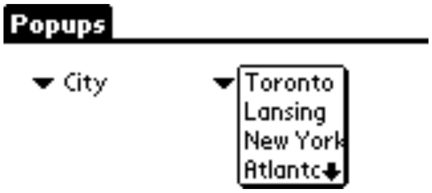
Grid – A rectangular display area that has vertical rows and horizontal columns. It can be used to display data created by a program or automatically loaded from a file. Grids are used to display data only. When a cell in grid is clicked on, a script is run. A grid can have a scrollbar. The cells in a grid can be text, numeric or checkboxes..



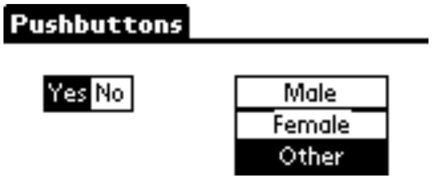
Label - a text string that is displayed on the display . Label objects always display the given text and are not alterable by the user (whereas a field object can have its value changed by the user). However, if the text is tapped, then the code for that label is executed. Label objects cannot be resized.



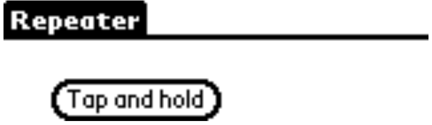
List - A list object displays a list of text strings with each string on a separate row of the list. If more rows are defined than can be displayed in the object's size, then scrollbars are drawn to allow the user to scroll through the entire list. Rows may be selected and become highlighted on the display by tapping on a row. If a list has associated code, then that code is executed when the user selects an item on the list. The width is adjustable, but the height depends on the number of lines.



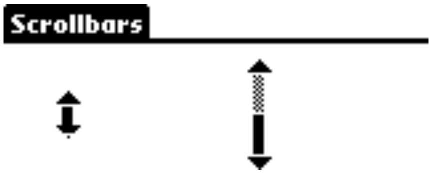
Popup Trigger - a text box preceded by a vertical arrow. When the arrow or popup text is tapped, a popup list of values for the item is displayed. A new value can be selected from the list of available values. If a new value is selected, then the code that is defined for the popup trigger is executed. This value is then displayed as the popup trigger text.
Example: a field may list the state description with a popup trigger adjacent to it. Clicking on the popup trigger causes the program to display a list of all state descriptions in a popup list from which the user can select a new state. The popup list disappears after the user selects a new value.



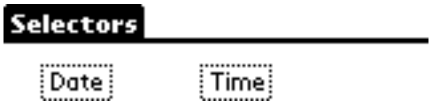
PushButton - a rectangle containing a text string. A PushButton may be "selected" in which case the text is displayed as inverted (black with white text) or "not selected" in which case the text is black on a white background. If the PushButton is tapped, then the code for that PushButton is executed.



Repeater - a repeater is similar to a Button except that the repeater will continue to fire the code associated with this object for as long as the object remains selected (as long as the user holds the clicks on the object). Normally this is used by displaying up or down or left and right arrow symbols from the symbol font as the Button's text. Depressing the up or down arrow can be programmed to increase or decrease the value of a variable and display the new value in a separate field on the form. Repeaters are used in the built-in applications frequently to continually increase a date's year, or month, or day values.



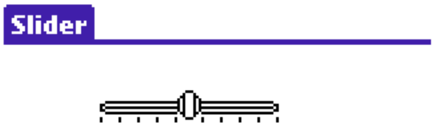
Scrollbar - a vertical bar that has arrows at each end. The user may tap anywhere on the bar or tap on the arrow at either end. If the scrollbar is tapped, the code for that scrollbar is executed.



Selector - an object that displays a text string enclosed in a gray rectangular frame. Normally used to show date or time. This is normally programmed so when the user taps on this object, the associated code is executed and it will bring up a date or time popup where the user can enter a new date or time. The new value is stored in this selector object's value and appears in the display on the form.

(not visible)

Shift Indicator - This indicator which looks like an up arrow is displayed if the device input mode is currently in upper case mode; otherwise nothing is displayed in the area reserved for this object.



Slider - an object which represents a value in a range between a minimum and maximum value.

03.L NS Basic/Symbian OS Built In Constants

The following constants are predefined and can be used within NS Basic/Symbian OS programs:

		default
NsbOn NsbOff	used with PushButton objects to set/determine status	1 0
NsbChecked NsbUnchecked	used with checkbox objects to set/determine status	1 0
NsbNormal NsbInverted NsbGrey NsbCustom	graphics commands pen constants	0 1 2 3
NsbKeyOrButton NsbPenDown NsbPenUp	getEvent return values	1 2 3
NsbWait NsbNoWait	Sound command parameter	0 1
NsbYes NsbNo	yes/no parameter	1 0
True False	boolean values	1 0

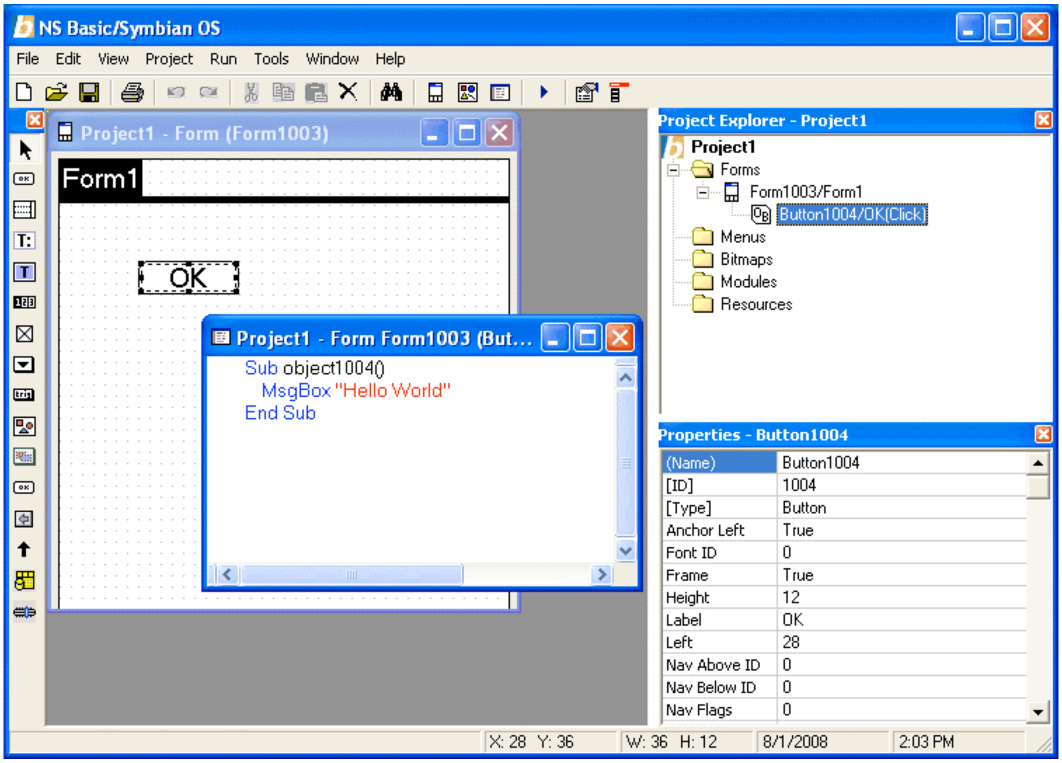
While these constants are required for some commands and functions, you can also use them yourself. For example, NS Basic/Symbian OS has no 'boolean' data type with yes/no or on/off properties. However, you can implement your own boolean operations as in the example shown below:

```
Dim mySwitch as integer
mySwitch=True

if mySwitch=True then
        ...
end if
```

04. Developing with the NS Basic/Symbian OS IDE

The NS Basic/Symbian OS IDE (Interactive Development Environment) is an easy to use tool to create NS Basic projects. The information you enter is a complete description of all the parts and code of your project that get saved in a .prj file. The IDE can then compile the file into a .prc file that is executable on a Device.



```
Sub object1004()
  MsgBox "Hello World"
End Sub
```

(Name)	Button1004
[ID]	1004
[Type]	Button
Anchor Left	True
Font ID	0
Frame	True
Height	12
Label	OK
Left	28
Nav Above ID	0
Nav Below ID	0
Nav Flags	0

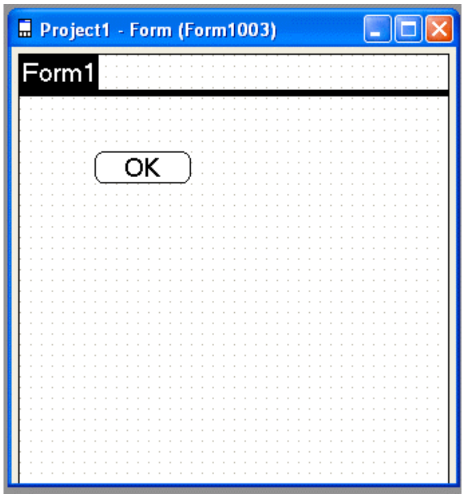
This is the main IDE screen, showing a typical configuration for a simple project. In this section, we'll look at each of the components of the above screen, what their purpose is and how they work.

04.A IDE Menu Options

File		
	New Project	Save and close the current project, then create a new project.
	Open Project...	Brings up the Open Project dialog screen. This can be used to start a new project, or open an existing one from the Project folder or from the list of recent projects. If the dialog is not cancelled, the current project is saved and closed before the new one is opened.
	Save Project	Save the current project. If it has not been saved yet, prompt for a file name.
	Save Project As...	Save a copy of the current project under a new name.
	Print Setup...	Set options for printing
	Print...	Print all code for the current project
	(recent projects)	Displays recent projects. Selecting one will open it after saving and closing the current project.
	Exit	Save and close the current project, then exit the NS Basic/Symbian OS IDE.
Edit		
	Undo	Undo the latest action
	Redo	Redo the latest action that was undone
	Cut	Cut to the clipboard
	Copy	Copy to the clipboard
	Paste	Paste from the clipboard
	Delete	Delete item
	Select All	Select all items
	Find	Find text in the code of the project
	Replace	Find and replace text in the code of the project
	Go to Line	Go to a line in a code window
View		
	Project Explorer	Open or hide the Project Explorer window.
	Properties Window	Open or hide the Properties window.
	Toolbox	Open or hide the Toolbox window.
	Toolbar	Display or hide the Toolbar window
	Status Bar	Display or hide the Status bar window
	Refresh	Redraw the IDE
Project		
	Add Form	Add a new form to the project.
	Add Menu	Add a new menu to the project.
	Add Bitmap	Add a bitmap, jpg or gif to the project
	Add New Module	Add a new code module to the project
	Add Existing Module...	Add an existing code module to the project.
	Add Resource	Adds an existing resource file to the project
	Startup Code	Add or edit the Project Startup Code
	Termination Code	Add or edit the Project Termination Code.
Format		
	Align	Align multiple selected objects on the Design Screen with each other. The options for Left, Centers, Right, Tops, Middles, Bottoms align objects with each other. Align to Grid aligns the top left corner of the object to the grid.
	Make the same size	Make multiple selected objects on the Design Screen the same size as the first object selected. The width, height

		or both can be duplicated.
	Size to Grid	Moves the object's top left corner to the nearest grid point, then resize the height and width to the nearest multiple of the grid size.
	Horizontal Spacing	*Equal*: Make the horizontal spacing between objects equal. *Increase*: Moves the second of two objects right. *Decrease*: Moves the second of two objects left. *Remove*: get rid of any horizontal space between two selected objects.
	Vertical Spacing	*Equal*: Make the vertical spacing between objects equal. *Increase*: Moves the second of two objects down. *Decrease*: Moves the second of two objects up. *Remove*: get rid of any vertical space between two selected objects.
	Center in Form	Centers the selected object vertically or horizontally on the form.
Run		
	Compile (F5)	Compile the project into an executable app that will run on a Device, using the setting in Compile/Download Options. As the project is compiling, the current module and final size will be displayed on the Status Bar.
	Run Installer	Will run the installer for the current app after it Is compiled.
Tools		
	Menu Editor	Brings up the menu editor
	Options	Allow various program options to be changed.
Window		
	Cascade	Cascade the currently open windows.
	Tile Horizontal	Tile the currently open windows horizontally.
	Tile Vertical	Tile the currently open windows vertically.
Help		
	Register...	Enter the serial number of your individual copy of NS Basic/Symbian OS. This will allow your applications to run without a time limit.
	Language Reference	Open up a standard Help file. This contains the Overview and Reference sections of the Handbook.
	NS Basic Website	Open a special page on the NS Basic website. This will give you news on the latest versions and other information from NS Basic/Symbian OS
	Tech Notes	Tech Notes with additional in depth information about NS Basic/Symbian
	Tutorials	Tutorials covering a number of different common programming tasks.
	About NS Basic/Symbian OS...	Display the current version information of NS Basic/Symbian OS.

04.B Design Screen



The Design Screen is a mockup of how your application will look when running on the device. To make it easier to see, it normally displays with a 2 times magnification, i.e. if the actual device screen is 160x160, then the Design Screen displays as 320x320. If you move an object, it will move in 2 pixel increments.

The background grid that displays by default does not show up on the actual device, but makes it easier to layout your form. You can change the spacing of the grid points or hide them by changing the setting in Options.

To add an object to the Design Screen, select it from the Toolbox and click in the Design Screen at the position you want the top left corner of your new object. The new object will display.

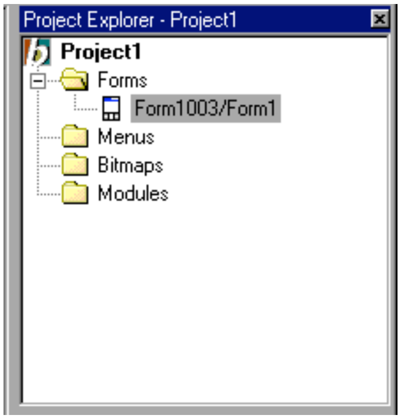
To select an object, click on it. Multiple objects can be selected by holding down the shift key while selecting. The functions under the Format menu can then be used to position objects relative to each other.

When an object is selected, its properties are displayed in the Properties Window. When you edit the properties there, the changes are reflected immediately on the Design Screen. To edit the code for an object, double click on it and a Code Window will open.

Clicking outside any of the objects will bring up the Properties Window for the form. Right clicking outside of any object allows you to edit the form's Before, After or Event Code and add new components to the project.

Clicking on the border will change the magnification of the screen from two times to one.

04.C Project Explorer

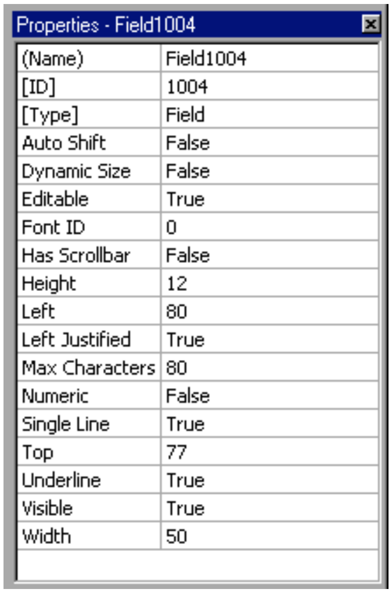


The Project Explorer is an easy way to navigate through your project and edit it. Your project is made up of Forms, Menus, Bitmaps and Modules. Of these, only a Form is required. If there is a box with a + sign in it to the left of an icon, clicking on the + sign will expand to show the list of items within that category. Similarly, clicking on a - sign will collapse the list.

Forms and Bitmaps are displayed with two names separated with a slash (/). The first name is the name of the form or object as used in your program; the second is its Title or Label. If the project, form or object has code, the word Before, After, Event, or Click will appear to the right of it.

Right clicking on any line in the Project Explorer opens up a menu which allows you to add, edit or delete information for that item.

04.D Properties Window



Properties - Field1004

(Name)	Field1004
[ID]	1004
[Type]	Field
Auto Shift	False
Dynamic Size	False
Editable	True
Font ID	0
Has Scrollbar	False
Height	12
Left	80
Left Justified	True
Max Characters	80
Numeric	False
Single Line	True
Top	77
Underline	True
Visible	True
Width	50

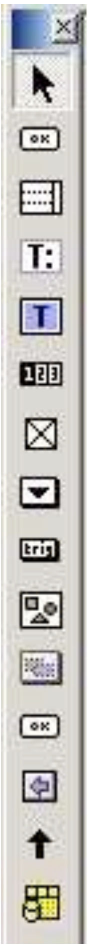
The Properties Window allows you to edit the properties of the projects, forms, objects, bitmaps and menus. When the Properties Window is open, selecting one of these items will cause the list of properties for it to be displayed.

Depending on the type of data, individual properties may bring up appropriate dialog boxes to make it easier to enter the correct data.

Not all properties are editable. The example above shows the properties of a Field object. The ID and Type are fixed and cannot be changed. Properties in square brackets are read only.

To cut and paste values of properties, use the right click menu in this windows. Use Ctl'C" and Ctl'V" to cut and paste objects.

04.E ToolBox



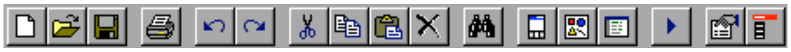
Use the ToolBox to select objects to be added to your project. Click on the object you want to add, then click again on the Design Screen at the top left corner of where you want the object to appear.

You can see the name of an object by holding the cursor over an object for a few seconds.

The objects are (in order of Toolbox)

Button
List
Label
Field
Pushbutton
Checkbox
Popup
Selector
Bitmap
Gadget
Repeating Button
Scrollbar
Shift Indicator
Grid
Slider

04.F ToolBar



The Toolbar is a handy way to get at some of the commonly used menu items. You can see the name of an icon by holding the cursor over the icon for a few seconds:

Create a new project
Open a project
Save the current project

Print the code of the current project

Undo the last action
Redo the action that was undone

Cut
Copy
Paste
Delete

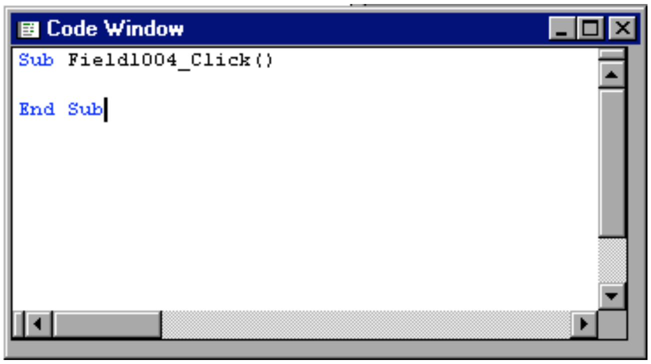
Search for text in the code

Add Form
Add Bitmap
Add Module

Compile

Open Properties Window
Open Menu Editor

04.G Code Window



```
Sub Field1004_Click()

End Sub
```

Whenever code is to be written, the Code Window appears. Multiple code windows can be opened simultaneously. The text is colored depending on its type:

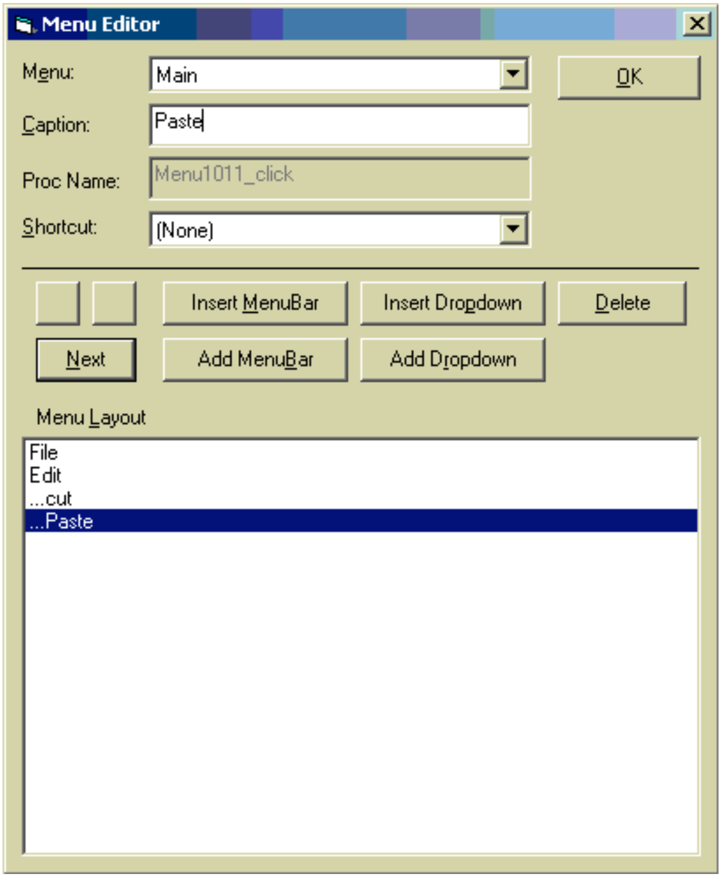
Black	Program text
Blue	NS Basic/Symbian OS keywords
Green	Comments
Orange	Operators
Purple	Strings
Yellow	Highlighted line

Right clicking inside the Code Windows brings up a few handy options. The Properties selection allows a wide range of features to be customized, including text coloring, tabs, font, keyboard shortcuts and window appearance. Advanced features include Comment and Uncomment for blocks of code.

Statements can be made longer than one line by using the "_" character at the end of a line.

Cut, Paste, Delete and other similar functions can be used from the Menu or the Toolbar.

04.H Menu Editor



Menu: Main

Caption: Paste

Proc Name: Menu1011_click

Shortcut: (None)

Menu Layout

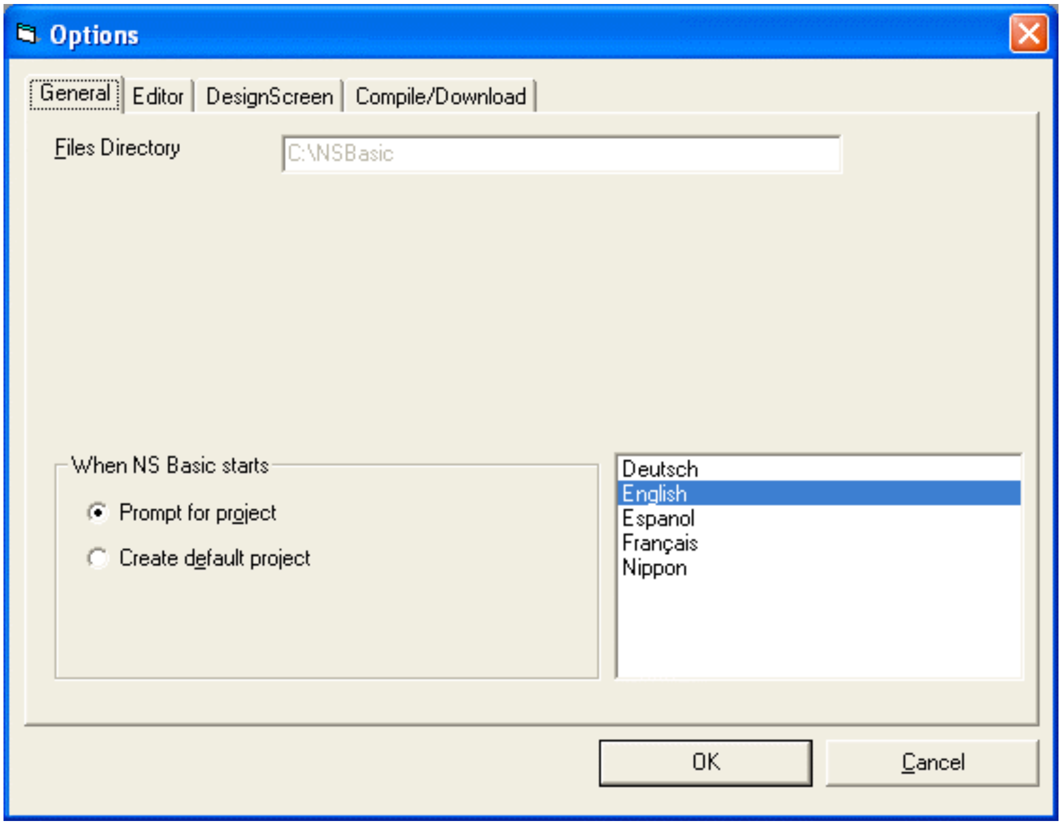
File
Edit
...cut
...Paste

The Menu Editor allows you to edit menus. Menus can also be edited in the Project Explorer.

Menu	The name of the menu
Caption	A field to edit the name highlighted in the Menu Layout section below.
Proc name	The name of the code segment to be executed when the menu item is selected by the user.
Shortcut	The character to display as a shortcut to the menu item
Insert Menubar	Add a new menubar item above the current selection
Add Menubar	Add a new menubar item below the current selection
Insert Dropdown	Add a new dropdown item above the current selection. Dropdown items start with "..." A "-" character will put a horizontal line in the dropdown list.
Add Dropdown	Add a new dropdown item below the current selection
Next	Move to the next item in the Menu Layout
Delete	Delete the current selected item in the Menu Layout
OK	Close and save the information

04.I Options

04.I.1 Options - General



Files Directory C:\NSBasic

When NS Basic starts

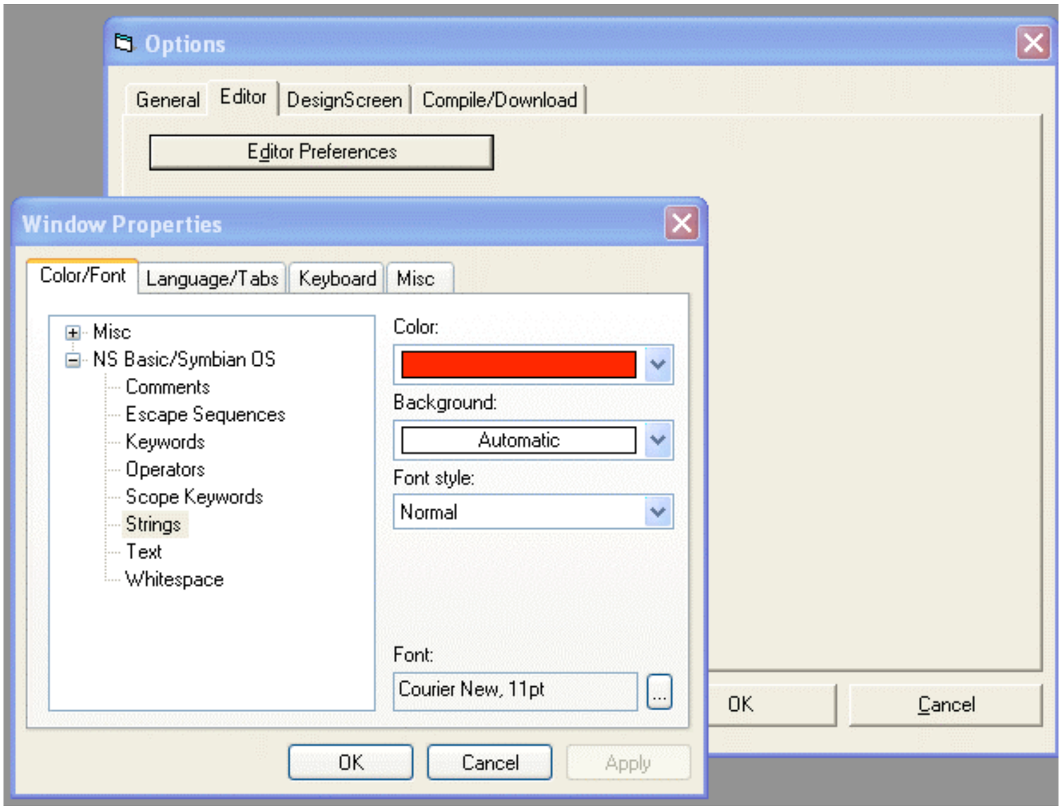
● Prompt for project

○ Create default project

Deutsch
English
Espanol
Français
Nippon

Files Directory	The path to the project. Normally, this is not changed once you have created a project.
When NS BASIC starts	If Prompt for project, the IDE will ask if you want to open a recent project or start a new one. Create default project will always start a new one.
(language)	The language of the IDE. Supported languages are in \program files\nsbasic\Symbian\lang. If you do not see your own language, you are free to translate StringTable_English into your own language, so long as you share it with us so we can give it to other users.

04.I.2 Options - Code Window



Color: 🟥

Background: Automatic

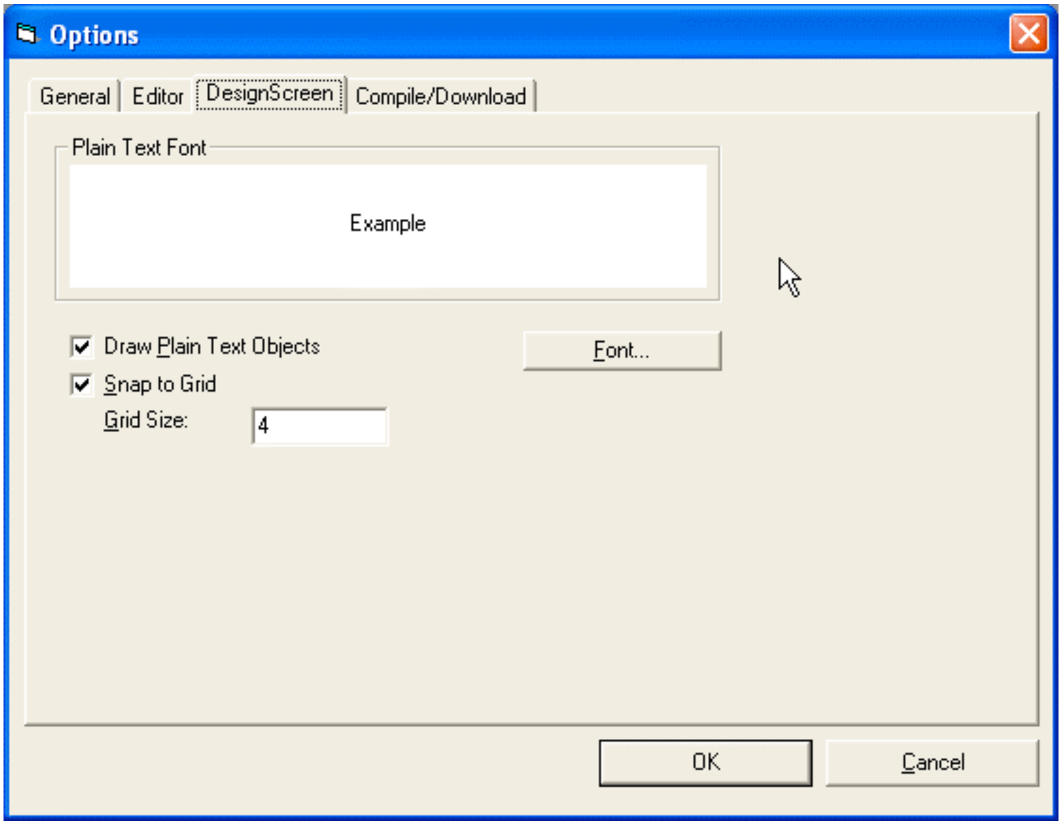
Font style: Normal

Font: Courier New, 11pt

The Code Windows Option allows a wide range of features to be customized, including text coloring, tabs, font, keyboard shortcuts and window appearance.

You can also call this screen up by right clicking within the Code Window.

04.I.3 Options - Design Screen



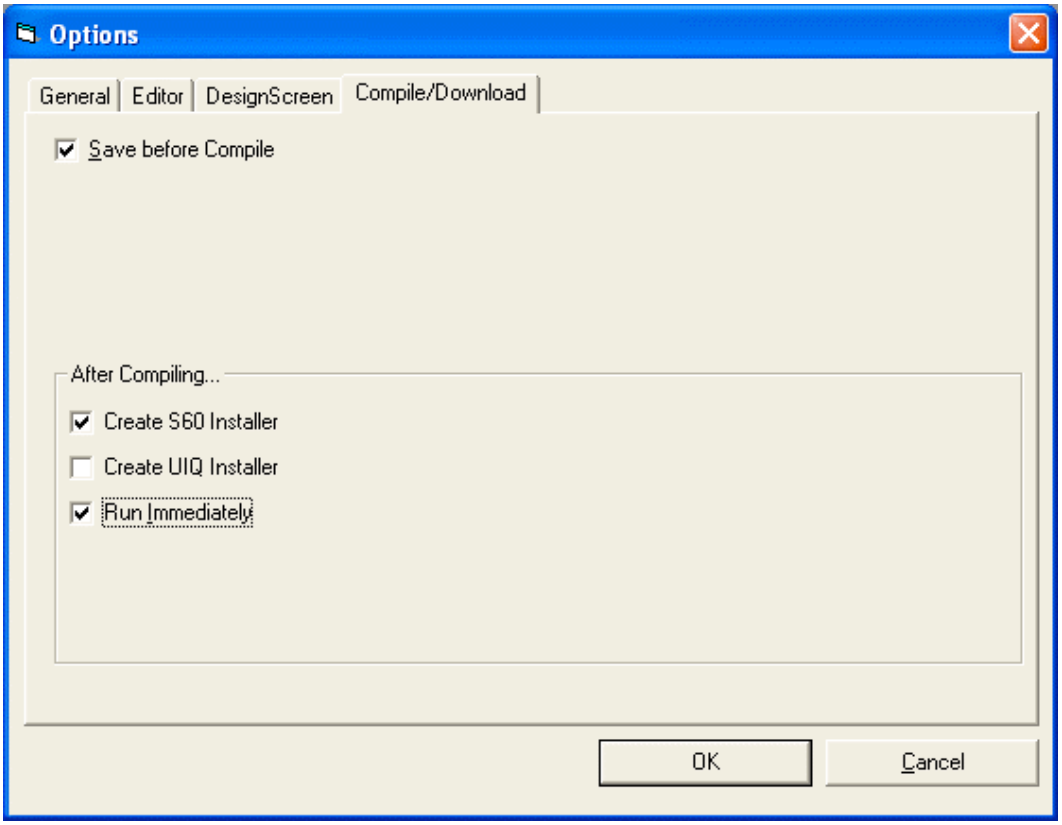
☑ Draw Plain Text Objects

☑ Snap to Grid

Grid Size: 4

Draw Plain Text Objects	Normally, objects are drawn on the Design Screen using a standard font. If you are using a font other than one of the ones that come with NS Basic/Symbian OS (for example, Chinese, Japanese, Thai, Greek, etc.) check this box. This same font is used in the Project Explorer and the Properties Windows to enter data which will appear on the Design Screen.
Font	Use this to specify the font to be used when Draw Plain Text Objects is checked.
Snap to Grid	Force object to align to grid when being placed or moved on Design Screen
Grid Size	The number of pixels separating the grid points.

04.I.4 Options - Compile/Download



☑ Save before Compile

After Compiling...

☑ Create S60 Installer

□ Create UIQ Installer

☑ Run Immediately

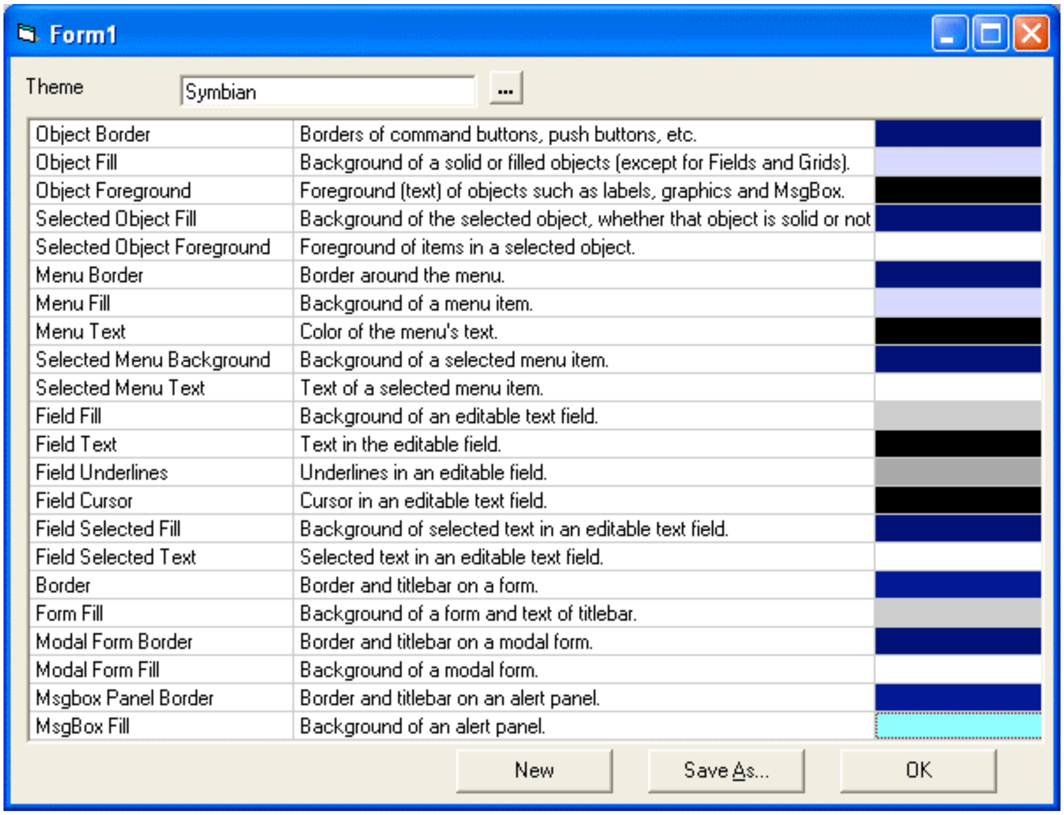
Save before Compile	A complete save is done of the project before each compile.
Create S60 Installer	Creates a .sisx file that can be used to install the app. This file will included everything to install a standalone app.
Create UIQ Installer	Creates a .sisx file that can be used to install the app. This file will included everything to install a standalone app.
Run Immediately	After the download is complete, start running the installer.

04.J Theme Editor

Themes allow you to enhance the appearance of your application by adding color to it. In Project Properties, you define a Theme. A theme defines colors for 22 different features of a form. There are 30 themes included NS Basic/Symbian OS.

To edit a theme or to create a new one, start the IDE and open a project. Select the Theme Editor under the Tools menu. It will automatically open the theme for the current form for editing, but allows you to also modify other themes or create new ones.

Projects are automatically given the "Symbian" theme. The theme you specify in Project Properties will be automatically be added as a resource to your project. If you use additional themes, you have to add them as resources. Themes are kept in c:\NSBasic_Symbian\Themes. You can put additional themes you create in that folder or elsewhere.



Theme: Symbian

Object Border	Borders of command buttons, push buttons, etc.	🟦
Object Fill	Background of a solid or filled objects (except for Fields and Grids).	⬜
Object Foreground	Foreground (text) of objects such as labels, graphics and MsgBox.	⬛
Selected Object Fill	Background of the selected object, whether that object is solid or not	🟦
Selected Object Foreground	Foreground of items in a selected object.	⬜
Menu Border	Border around the menu.	🟦
Menu Fill	Background of a menu item.	⬜
Menu Text	Color of the menu's text.	⬛
Selected Menu Background	Background of a selected menu item.	🟦
Selected Menu Text	Text of a selected menu item.	⬜
Field Fill	Background of an editable text field.	▨
Field Text	Text in the editable field.	⬛
Field Underlines	Underlines in an editable field.	▨
Field Cursor	Cursor in an editable text field.	⬛
Field Selected Fill	Background of selected text in an editable text field.	🟦
Field Selected Text	Selected text in an editable text field.	⬜
Border	Border and titlebar on a form.	🟦
Form Fill	Background of a form and text of titlebar.	▨
Modal Form Border	Border and titlebar on a modal form.	🟦
Modal Form Fill	Background of a modal form.	⬜
Msgbox Panel Border	Border and titlebar on an alert panel.	🟦
MsgBox Fill	Background of an alert panel.	🟦

Theme	The name of the theme. Themes are stored in \NS Basic_Symbian\Themes.
(item name)	The class of the theme item. All items in the project will use the color that is specified for the item name.
(description)	The description of what the item name describes.

05. Commands Overview

05.A Arithmetic Commands

Let	calculates the value of an expression and stores the result in a variable

05.B Control and Logic Commands

If <test> Then Else ElseIf End if	tests an expression and executes statements depending on the result of the test
Do [Until|While] Loop Exit Do	repeats a series of statements until some test ends the execution
For Next Exit For	repeats execution of a series of statements and increments a control variable on each execution
GoTo	transfers control to a section of statements identified by a label
Gosub Return	transfers control to a section of code and returns
NextForm	Changes the displayed form to a different form.

Select Case
   Case
   Case Else
End Select
	tests a variable's value and executes selected statements depending on the value

Stop	Stops program execution

05.C Module Definition and Control Commands

Sub	defines the start of a subroutine
Function	defines the start of a function
End Sub	defines the end of a subroutine
End Function	defines the end of a function
Call	transfers control to another subroutine
Exit Sub	exits the subroutine
Exit Function	exits the function

05.D Graphics Commands

`CreateWindow`	Creates a new window in which graphics can be displayed
`DestroyWindow`	Destroys a previously created graphics windows
`DrawBitmap`	Draws a bitmap on the graphics display
`DrawChars`	Draws text strings on the graphics display
`DrawLine`	Draws a line on the graphics display
`DrawRectangle`	Draws a rectangle on the graphics display
`EraseWindow`	Erases the graphics display window
`FillRectangle`	Fills a rectangle with a user defined color
`SetCurrentWindow`	Sets the current graphics window to which graphics commands apply.
`SetTheme`	Sets the color theme for objects and the background.

05.E Miscellaneous Commands

`Beep`	Sounds an audible beep sound
`Chain`	Terminates the current program and launches a new Device program
`Cursor`	Moves the screen cursor to a given screen coordinate
`Delay`	Delays program execution for a specified amount of time and places the program into a wait state until the time expires.
`Dim`	Defines data variables, arrays, or files that are used in the code.
`Display`	Displays the values of one or more variables at the current screen location
`Global`	Defines data variables that are globally used throughout all subroutines and functions
`MenuDraw`	Draws a specified menu at the top of the display screen
`MenuErase`	Erases the current menu from the display screen
`MenuReset`	Cancels any dropdown sub-menu that has previously been selected.
`Msgbox`	Displays a message box on the display screen that does not give the user any options to respond to. If a user response is desired, use ALERT.
`SetEventHandled`	Tells the operating system to ignore a special event
`Sound`	Plays a sound of a given frequency and duration through the device speaker.

06. Functions Overview

06.A Arithmetic Functions

`Abs`	Returns the absolute value of an arithmetic expression
`Cbrt`	Returns the cube root of a number
`Ceiling`	Returns the next higher integer number than a given arithmetic expression
`Exp`	Returns the exponential of a number
`Floor`	Returns the next lower integer number than a given arithmetic expression
`Int`	Returns the integer value of an arithmetic expression
`Log`	Returns the natural log of a number
`Log10`	Returns the base 10 log of a number
`Mod`	Returns the remainder of one arithmetic expression divided by another
`Pow`	Returns the result of one number raised to the power of a second number
`Power10`	Returns the result of raising a number to the specified power of 10
`Rem`	Returns the remainder of dividing one number by another number
`Round`	Rounds the results of an arithmetic expression up to the nearest fractional digit
`Sign`	Returns an indicator of the sign (positive or negative) of an arithmetic expression
`Sqrt`	Returns the square root of an arithmetic expression
`Trunc`	Returns a float with decimal places truncated

06.B Trigonometric Built-In Functions

`Acos`	Returns the arc-cosine of an angle.
`Asin`	Returns the arc-sine of an angle.
`Atan`	Returns the arc-tangent of an angle.
`Atan2`	Returns the arc-tangent of an angle expressed as the division of two numbers
`Cos`	Returns the cosine of an angle.
`Sin`	Returns the sine of an angle.
`Tan`	Returns the tangent of an angle.
`Acosh`	Returns the hyperbolic arc-cosine of an angle.
`Asinh`	Returns the hyperbolic arc-sine of an angle.
`Atanh`	Returns the hyperbolic arc-tangent of an angle.
`Cosh`	Returns the hyperbolic cosine of an angle.
`Sinh`	Returns the hyperbolic sine of an angle.
`Tanh`	Returns the hyperbolic tangent of an angle
`DegToRadians`	Returns the radians of angle input in degrees
`RadToDegrees`	Returns the degrees of an angle input in radians

06.C String Manipulation Built-In Functions

`Asc`	Returns the ASCII value (0 to 255) of the leftmost character of the string
`Chr`	Returns a one-character string representing the character whoseASCII value is given.
`Format`	Formats a numeric value into a string of text
`InStr`	Checks if one string is contained within another string
`LCase`	Returns a string that is a lower case conversion of the input string
`Left`	Returns the specified number of characters on the left side of a string
`LeftPad`	Returns the input string padded on the left with spaces to make the string a given length
`Len`	Returns the length of a string
`Ltrim`	Eliminates leading spaces from a string
`Mid`	Returns a portion of the input string
`Proper`	Returns a string with the 1st letter of each word capitalized
`Right`	Returns the specified number of characters from the right side of a string

RightPad	Returns the input string padded on the right with spaces to make the string a given length
Rtrim	Eliminates trailing spaces from a string.
Str	Returns a string representation of the value of an arithmetic expression
TestNum	Tests if a string represents a valid numeric value.
Trim	Eliminates both leading and trailing spaces from a string.
Ucase	Returns a string that is the upper case conversion of the input string
Val	Returns the numeric value of a string

06.D Date Manipulation Built-In Functions

AddDays	Returns a date by adding the given number of days to the input date
AddMonths	Returns a date by adding the given number of months to the input date
AddYears	Returns a date by adding the given number of years to the input date
DateDiff	Returns the number of days between two dates
DateVal	Returns a date value from the input numeric variables of year, month, and day
DateMMDDYY	Returns a string of the format "MM/DD/YYYY" from the input date
Day	Returns the day from the input date
DayOfWeek	Returns the day of the week (1-7) from the input date
DayOfYear	Returns the day of the year (1-366) from the input date
FirstOfMonth	Returns a date which is the first day of the month of the input date
LastOfMonth	Returns a date which is the last day of the month of the input date
MMDDYYToDate	Returns a date value from the input string in the format "MM/DD/YYYY"
Month	Returns the month from the input date
MonthDay	Returns a string of the format "MM/DD" from the input date
PopupDate	Prompts the user for a date using the standard date request popup
SubtractDays	Returns a date by subtracting the given number of days from the input date
SubtractMonths	Returns a date by subtracting the given number of months from the input date
SubtractYears	Returns a date by subtracting the given number of years from the input date
ToDate	Returns a date value from the input string which is in the format "YYYY/MM/DD"
Today	Returns a date from the computer which is today's date
Year	Returns the year from the specified date
YearMonth	Returns a string of the format "YY/MM" from the specified date

06.E Time Manipulation Built-In Functions

Hour	Returns the hours from the input time
HourMin	Returns a string of the format "HH:MM" from the input time
HourMinAMPM	Returns a string of the format "HH:MM XX" where XX is AM or PM
Minute	Returns the minutes from the input time
Now	Returns the current time from the computer's internal clock
PopUpTime	Prompts the user for start and end times using the standard time request popup
Second	Returns the seconds from the input time
TimeDiff	Returns the number of seconds between two times
TimeVal	Returns a time from the input numeric variables of hour, minute, and second
ToTime	Returns a time from the input string which is in the format "HH:MM:SS"

06.F Miscellaneous Built-In Functions

Alert	Returns the user response to an alert message
AppLaunch	Call another app from within NS Basic
GetEventType	Returns the type of event that caused the special event code to be executed
GetKey	Returns the last key or button pressed
GetPen	Returns the current pen position and whether the pen is up/down.
LoadLibrary	Prepare a Library for use
NoOccurs	Returns the maximum subscript for an array variable

Rand	Returns a random number between 0.0 and 1.0
SysEventAvailable	Checks if there are any pending system events
SysInfo	Returns various system information
SysInfoSet	Sets system information
SysTrapFunc	Call a Symbian OS function as a function
SysTrapSub	Call a Symbian OS function as a subroutine

06.G File Built-In Functions

		No Key	Key
DbClose	Closes an open file.	•	•
DbCreate	Creates a file on the Device.	•	•
DbCreateDatabasefromResource	Creates a file from resource contained in the project		
DbDelete	Deletes a file record by key		•
DbErase	Removes a file from the Device.	•	•
DbFind	Finds a file record by key		•
DbGet	Reads values from the current file record	•	
DbGetNoRecs	Returns the number of records in a file	•	•
DbInsert	Inserts a new record in a file by key		•
DbOpen	Opens a file and initializes it for processing	•	•
DbPosition	Locates a record by relative record number	•	
DbPut	Writes values to the current record	•	
DbRead	Reads a record by key		•
DbReadNext	Reads the next record		•
DbReadPrev	Reads the previous record		•
DbReset	Resets a file to the beginning record.		•
DbUpdate	Updates the contents of a record by key		•

06.H Serial I/O Built-In Functions

SerialOpen	Opens the serial port and prepares for input/output
SerialClose	Closes the serial port to discontinue its use
SerialReceive	Accepts input from the serial port
SerialReceiveWithEvent	Sets serial communications to wait for input from serial port.
SerialSend	Transmits data out through the serial port
SerialSet	Sets the value of several serial port parameters to control transmission options

07. NS Basic/Symbian OS Reference

The Reference chapter contains an entry for every Statement, Function and Object used in NS Basic/Symbian OS. The entries are listed in the index under Statement, Function or Object.

Abs Function

ABS(*theNumber* as float)

Description
Returns a float with the absolute value of the argument.

Example
```
y=-4
x=Abs(y) will result in x=4

or z=Abs(balance*rate)
```

Acos Function

ACOS(*theNumber* as float)

Description
Returns a float with the arc-cosine of the arithmetic expression argument. The result angle is expressed in radians

Example
```
y=DegToRadians(180)
x=Acos(y) will result in the arc-cos of 180 degrees
```

Acosh Function

ACOSH(*theNumber* as float)

Description
Returns a float with the hyperbolic arc-cosine of the arithmetic expression argument. The result angle is expressed in radians.

Example
```
y=0.707106
x=Acosh(y) 'will result in 0.7853992681 radians, that is 45 degrees
```

AddDays Function

ADDDAYS(*theDate* as Date, *Days* as Integer)

Description
Adds *days* to *theDate* and returns the new date value.

theDate should be a valid date value.

Example
```
Dim theDate as Date
Dim newDate as Date
TheDate=ToDate("03/01/98")
newDate=AddDays(theDate, 45) 'adds 45 days

newDate will now have a date of April 14, 1998
```

AddMonths Function

ADDMONTHS(*theDate* as Date, *Months* as Integer)

Description
Adds *Months* to *theDate* and returns the new date value.

Example
```
Dim theDate as Date
Dim newDate as Date
TheDate=ToDate("03/01/98")
newDate=AddMonths(theDate, 15) 'adds 15 months

newDate will now have a date of June 1, 1999
```

AddYears Function

ADDYEARS(*theDate* as Date, *Years* as Integer)

Description
Adds *Years* to theDate and returns the new date value.

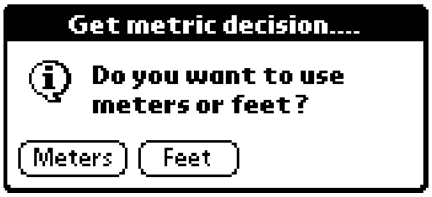
Example
```
Dim theDate as Date
Dim newDate as Date
TheDate=ToDate("03/01/98")
newDate=AddYears(theDate, 4) 'adds 4 years

newDate will now have a date of March 1 , 2002
```

Alert Function

ALERT(*title* as String, *msg* as String, *type* as Integer, *button0* as String, *button1* as String, ... , *ButtonN* as String)

Description
Displays a Device alert box of type '*type'* and waits for the user to select one of the buttons. Returns an integer with the button selected, starting from zero.


Get metric decision....

Do you want to use meters or feet?

Type is the format of the alert box. *title* is a text string that will be displayed at the top of the alert box. *Msg* is the text of the information, confirmation, warning, or error message that displays in the middle of the alert box. *Button*, *button1,* etc. are the texts to appear in each button at the bottom of the alert box. You may provide as many buttons as will fit, but generally 2 or 3 are used.

Type=0	display an 'information' type alert box
1	display a 'confirmation' type alert box
2	display a 'warning' type alert box
3	display an 'error' type alert box

Example
```
Dim result as Integer
Result=Alert("Get metric decision.......", "Do you want to use meters or
feet?",0,"Meters","Feet")
If result=0 then
  MsgBox "Meters"
else
  MsgBox "Feet"
end if
```

AppLaunch Function

APPLAUNCH(*cardNo* as integer, *pgm* as string, *cmd* as integer, *data* as string)

Description
Returns an integer with result code from the call. *CardNo* should be 0, *pgm* is the name of the program to launch, *cmd* is a command to be passed to *pgm* and *data* is string with data to be passed to *pgm*. This function launches another app from NS Basic/Symbian OS. Unlike CHAIN, execution continues in NS Basic/Symbian OS after pgm returns.

Example
```
' in an object's code:
PalmPrint "Hello World"  'this calls the PalmPrint routine in a code
...
sub PalmPrint(PrintData as string)
   dim res as integer
   res=AppLaunch(0, "PalmPrint", 32768, PrintData)
end sub
'This sample calls PalmPrint from Steven's Creek (http://www.stevenscreek.com) to
print "Hello World" to an IR printer.
```

Asc Function

ASC(*theString* as String)

Description
Returns an integer with the ASCII number for the leftmost character of *theString.* Will be in range 0-255.

Example
```
Dim s as String
Dim i as Integer
s="a"
i=Asc(s) 'i would have a value of 97
```

or
```
Dim s as String
Dim i(5) as Integer
Dim j as Integer
s="Hello"
for i=1 to 5
  i(j)=Asc(Mid(s,i,1))
next
This will put theASCII numeric values for "Hello" into the array "i" as follows
i(1)=72 'ASCII value for 'H'
i(2)=101 'ASCII value for 'e'
etc. for i(3),i(4), and i(5)
```

Asin Function

ASIN(*theNumber* as float)

Description
Returns a float with the arc-sine of the arithmetic expression argument. The result angle is expressed in radians.

Example
```
y=DegToRadians(180)
x=Asin(y) 'will result in the arc-sine of 180 degrees
```

Asinh Function

ASINH(*theNumber* as float)

Description
Returns a float with the hyperbolic arc-sine of the arithmetic expression argument. The result angle is expressed in radians.

Example
```
y=DegToRadians(180)
x=Asinh(y) 'will result in the hyperbolic arc-sine of 180 degrees
```

Atan Function

ATAN(*theNumber* as float)

Description
Returns a float with the arc-tangent of the arithmetic expression argument. The result angle is expressed in radians.

Example
```
y=DegToRadians(180)
x=Atan(y) 'will result in the arc-tangent of 180 degrees
```

Atan2 Function

ATAN2(*x* as float, *y* as float)

Description
Returns a float with the arc-tangent of the arithmetic expression of *x* divided by *y.* The result of *x*/*y* must be an angle in radians.

Example

```
x=Atan2(x, y) 'will result in the arc-tangent of the angle of x/y
```

Atanh Function

ATANH(*theNumber* as float)

Description
Returns a float with the hyperbolic arc-tangent of the arithmetic expression argument. The angle must be expressed in radians.

Example
```
y=DegToRadians(180)
x=Atanh(y) 'will result in the hyperbolic arc-tangent of 180 degrees
```

Beep Statement

BEEP

Description
Plays a short audible beep sound.

Returns: Nothing.

Example
```
If not testNum(inpField, " ",3,2)=0 then
  Beep
  Msgbox "Input value is not numeric."
end if
```

Bitmap Object

BITMAPS are powerful objects that allow images to be displayed and can execute code when they are tapped on. The image to be displayed is set at design time. A Bitmap ID (stored in the Bitmaps folder in the Project Explorer) can only be used in a single Bitmap object.

Gradient Buttons and Labels: You can create objects that have a background that gradually goes from one color at the top to another at the bottom. At compile time, the image is transferred into a bitmap resource in the project, and saved as a file in \NSBasic_Symbian\bitmaps\<projectName>.

The properties which control gradients are:

Gradient Color1: The top (or left) color.
Gradient Color2: The bottom (or right) color
Gradient Style: 1 for top to bottom, 0 for left to right.

You can use a Bitmap instead of a Label object of the same size. Since Bitmap objects can have a script (just like a Button object), BitMap object buttons work the same way as Button object buttons - but there are more options on how they can appear.

Caption and Fonts: Bitmaps can have text on them, using any font that is on your desktop system. The properties are:

Caption: The text to appear
Alignment: 0 for left, 1 for right, 2 for centered.
Appearance: 0 for flat, 1 for 3D
Border Style: 0 for none, 1 for 1 border around the image.
Caption Color: The color of the text.
Font Name: The name of the font. Can use any font installed on your system.
FontSize: The size of the letters.
FontBold: Are the characters in boldface?
FontItalic: Are the characters italicized?
FontStrikeThru: Is there a line through the characters?
FontUnderline: Is there a line under the characters?

Images on Bitmaps: You can specify the image to appear on a bitmap as well as its size. A Bitmap can use either Gradients, Captions and Fonts or an Image – not both. The properties are:

Picture: The pathname to the image. It can be bmp, gif, jpg and other format.
Stretch: If False, the image will be the size of the original image, aligned at the top left. If the image if larger than the bitmap object, it will be clipped. If Stretch is True, the image will be scaled to fit the current size of the bitmap object.

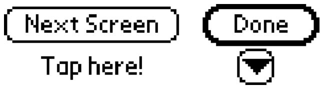
The maximum size of a bitmap's image is 65,512 bytes. The formula for calculating the size of an image is (width * height) * 8.

Properties Supported (Set at design time)
Alignment, Appearance, Bitmap ID, BorderStyle, Caption, Caption Color, Font Name, Font Bold, Font Italic, Font Size, Font Strikthru, Font Underline, Gradient Color 1, Gradient Color 2, Gradiant Style, Picture, Stretch, Visible

Methods Supported (See "Methods")
Hide, ID, Show, Index, Type, Left, Top

Button Object

A BUTTON is a rectangular box that can contain text. When the user taps on the button, the program code associated with that button is executed.



Properties Supported (Set at design time)
Left, Top, Width, Height, Label, Font ID, Anchor Left, Frame, Non-bold Frame, Visible

Methods Supported (See "Methods")
Hide, Show, Redraw, Label, ID, Index, Type

Call Statement

CALL *subName*(*argList*)

Description
Transfer control to another subroutine. The *argList* contains a list of arguments separated by commas that are parameters passed to the called *subName.* Arguments may be used to pass data values to the called program or to specify the names of variables that are to be given new values by the called program.

Arguments in the list may be an expression if the argument is passing data to the called program. The expression will be calculated and the resulting value will be passed to the called routine.

There must be a one-to-one correspondence between the number of arguments in *argList* and the number of arguments the called subroutine is expecting as defined in its SUB definition statement. Arguments should also be the same type(string, float,etc) as the called program is expecting. The names of arguments in *argList* does not have to be the same as the names of arguments defined in the called program because arguments are matched up by position in the calling *argList*.

Subroutines cannot be inside another subroutine or function.

Example

```
  ...
  Call PayrollRoutine(Salary, Deductions, NoDependents, NetSalary)
End Sub

Sub PayrollRoutine(Sal as Float, Deduct as Float, NoDepends as Integer, Net as
Float)
  Let Net=Sal-Deduct
End Sub
```

Ceiling Function

CEILING(*theNumber* as Float)

Returns: an integer number

Description
Returns an integer by rounding *theNumber* to the next higher integer. Negative numbers are rounded to the next higher negative number.

Example
```
Dim x as float
Dim result as integer
x=-33.2
result=ceiling(x) 'will calculate result of -33
```

Chain Statement

CHAIN 0, *ProgramName* as string

Description
Transfers control another app in your program's directory. This could be another NS Basic program or any StyleTap compatible app. *ProgramName* is the name of the new program to transfer to. Program names are case sensitive. To install other apps in your program's directory, add them as resources to your project. Do not use a .prc extension in *ProgramName.*

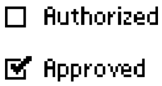
Example
```
chain 0, "Claims"
```

CheckBox Object

A CHECKBOX object is a text field preceded by a square box that may or may not have a check mark inside the square box. The user causes a checkbox to be "checked" or "not checked" by tapping inside the box. Checkbox objects are normally used to indicate a choice of some parameter or value where the possible choices are either yes/no, on/off, etc. Use the Group ID property to make checkboxes mutually exclusive.

Properties Supported (Set at design time)
Label, Font ID, Text, Group ID, Anchor Left, Selected, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Redraw, Text, Status, ID, Index, Type



Example
```
Msgbox "The checkbox status is " + str(chkYesNo.status)
If mydb_yes_no_fld=1 then chkYesNo.status = nsbChecked else chkYesNo.status =
nsbUnchecked
```

Chr Function

CHR(*x* as integer)

Description
Returns a 1 character string which is the ASCII character whose ASCII value is the integer *x*. *x* must have a value from 0 to 255.

Example
```
dim s as string
S=chr(13) 'creates a string whose 1st and only character is carriage return.
```

Controls Function

CONTROLS(*index* as integer).*propertyName*

Description
References a property or method of an object by its *index* position on a form. Controls(0) is the title bar of the form(if it has one). The objects of the form are indexed sequentially. Controls can be used to reference any property or event that is applicable to the object. The Type property is handy to determine the type of an object in the Control array.

Example
```
i=Button1004.index
controls(i).text="New button name"
```

Cos Function

COS(*theAngle* as float)

Description
Returns a float with the cosine of the arithmetic expression argument. The angle must be expressed in radians.

Example
```
y=DegToRadians(180)
x=Cos(y) 'will result in the cosine of 180 degrees
```

Cosh Function

COSH(*theAngle* as float)

Description
Returns a float with the hyperbolic cosine of the arithmetic expression argument. The angle must be expressed in radians.

Example
```
y=DegToRadians(180)
x=Cosh(y) 'will result in the hyperbolic cosine of 180 degrees
```

CreateWindow Statement

CREATEWINDOW(*windowName* as string, *xStart* as Integer, *yStart* as Integer, *width* as Integer, *height* as Integer)

Description
Create a new separate graphics window to be used for subsequent graphic commands. It's better to use this in the Form After code of a form than the Form Before code.

Example
```
CreateWindow("graphWin", 10, 20, 50, 100)
```

Cursor Statement

CURSOR *x* , *y*

Description
Moves the screen cursor(also called insertion point) to the *x* and *y* screen coordinates. Use to set position before DISPLAY.

Example
```
Cursor 20, 45
```

DateDiff Function

DATEDIFF(*dateVar1* as Date, *dateVar2* as Date)

Description
Returns the number of days between *dateVar1* and *Datevar2*.

Example
```
Dim dateVar1 as Date
Dim dateVar2 as Date
Dim result as integer
DateVar1=ToDate("2000/05/04")
DateVar2=Today()
Result=DateDiff(dateVar1, dateVar2)
```

DateMMDDYY Function

DATEMMDDYY(*theDate*)

Description
Returns a string of the form MM/DD/YYYY converted from *theDate.*

Example
```
Dim theDate as Date
Dim result as String
TheDate=DateVal(1995,6,23)
Result=DateMMDDYY(theDate) 'will give the string "06/23/1995"
```

DateVal Function

```
DATEVAL(theYear as Integer, theMonth as Integer, theDay as Integer)
```

Description
Returns a Date type by converting *theYear*, *theMonth*, and *theDay* inputs to Date format. Year values may be 2 or 4 digits(e.g. 95 and 1995 are equivalent.)

Example

```
Dim result as Date
Result=DateVal(1995,11,17) 'will set result to the date of 11/17/1995
```

Day Function

DAY(*theDate* as Date)

Description
Returns an integer which is the day value from the Date variable *theDate.*

Example

```
Dim theDate as Date
Dim result as Integer
theDate=DateVal(1995,6,23)
result=Day(theDate) 'will calculate result=23
```

DayOfWeek Function

DAYOFWEEK(*theDate* as Date)

Description
Returns an integer from 1=Sunday to 7=Saturday.

Example

```
Dim theDate as Date
Dim result as Integer
theDate=DateVal(1998,9,27)
result=DayOfWeek(theDate) 'will calculate result=1 because the date is a Sunday
```

DayOfYear Function

DAYOFYEAR(*theDate* as Date)

Description
Returns an integer from 1 to 366 which is the day of the year that *theDate* represents.

Example
```
Dim theDate as Date
Dim result as Integer
theDate=DateVal(1999,2,15)
result=DayOfYear(theDate) 'will calculate result=46th day of the year
```

DbClose Function

DBCLOSE(*dbName* as FileID)

Description
Closes a file and makes it no longer available for actions. Returns an integer error code. See DBOPEN for a table of result codes.

Example
```
Dim err as Integer
Err=DbClose(CustomerDb)
```

DbCreate Function

DBCREATE(*dbName* as FileID, *fileName* as String, *NotUsed* as Integer, *UID3* as String)

Description
Creates a new file on a specified memory card of the Device, and returns an integer result. See DBOPEN for a table of result codes.

DbName references the file inside the program. All actions on the file will identify it by this name. *FileName* is the name of the file in the device's memory. *NotUsed* is always 0. *UID3* is the 8 digit UID3 which can be used to identify the owner or creator of the file. Normally, you will use the same UID3 as you used for your project. To avoid duplications, Symbian allows you to register your UID3 on their website to insure it is unique and not duplicated by someone else. The naming convention is to follow your file name with a hyphen and your UID3. You may leave this blank if you are not concerned with conflicts.

Example
```
Dim result as integer
Dim CustomerDb as database
Result=DbCreate(CustomerDb, "Company-Test", 0, "E0000001")
```

DbCreateDatabaseFromResource Function

DBCREATEDATABASEFROMRESOURCE(*ResourceType* as String, *ResourceID* as Integer)

Description
(Obsolete – included for compatibility with older apps)
This function allows you to extract files and Libraries from your prc file and install them on your device, just as if they had been transferred by Hotsync.

To add a resource to a project, use Add Resource from Project menu, or right click in the Project Explorer. Select the desktop file that you want to add as a resource: .pdb or .prc files can be added. *ResourceID* is generated automatically by the system. *ResourceType* defaults to DBIM, but can be set to anything you like. The maximum size of a resource is 64k, so your prc or pdb cannot be larger than that.

At runtime, you can call DbCreateDatabaseFromResource at any point to extract the Library or file. The function returns an integer result: see the table under DBOPEN for the meanings of the messages.

Example
```
Dim res as integer
res=dbCreateDatabaseFromResource("DBIM",1015)
```

DbDelete Function

DBDELETE(*dbName* as FileID, *dbKey* as anyVarType)

Description
To delete a record given its record key. The file must have been previously opened with `DbOpen.` Returns an integer result. See DBOPEN for a table of result codes.

Example
```
Dim result as integer
Result=DbDelete(CustomerDb, "123") 'deletes customer record with key "123"
```

DbErase Function

DBERASE(*dbName* as FileID)

Description
Delete the file from the device's memory.. The file will be permanently erased and no longer accessible. A file may not be deleted unless it has been opened at least once. The file must be closed. Returns an integer result. See DBOPEN for a table of result codes.

Example
```
Dim res as Integer
Res=DbErase(CustomerDb)
```

DbFind Function

DBFIND(*dbName* as FileID, *dbKey* as anyVarType)

Description
To determine if the record exists in the file with the given key value. The file must have been previously opened with DbOpen. *dbkey* must be a variable: it cannot be an expression. Returns an integer result. See DBOPEN for a table of result codes.

Example
```
Dim err as Integer
Err=DbFind(CustomerDb, "362")
If err>0 then
  Msgbox "Can't find customer 362's record"
end if
```

DbGet Function

DBGET(*dbName* as FileID, *aVariables* as varList)

Description
To read data from the current record and current offset in the file. The variables in *aVariables* may be any of the valid variable types (String, float, etc.) The file must have been previously opened with DBOPEN. Returns an integer result. At end of file, -1 is returned. See DBOPEN for a table of result codes.

Example
```
Dim res as Integer
Res=DbGet(OrdersDb, OrderNo)
```

DbGetNoRecs Function

DBGETNORECS(*dbName* as FileID)

Description
Returns the number of records in a file. The file must have been previously opened with DBOPEN. Returns an integer result.

Example
```
Dim res as Integer
Res=DbGetNoRecs(OrdersDb) 'res has the number of records
If res>0 then
  res=dbPosition(OrdersDb, res, 0) 'position to last record
  res=dbGet(OrdersDb, OrderKey) 'order key has highest key in file
end if
```

DbInsert Function

DBINSERT(*dbName* as FileID, *dbKey* as anyVarType, *varlist*)

Description
To insert a new record into the file with the given key. All of the variables in *varlist* are written into the new record. Records cannot exist that duplicate a key, so insert of a record with a key that already exists returns an error. *dbKey* must be a variable: it cannot be an expression. The file must have been previously opened with DBOPEN. Returns an integer result. See DBOPEN for a table of result codes.

Example
```
Dim res as Integer
Res=DbInsert(CustomerDb, "998", custName, custAddress, custPhone)
```

DbOpen Function

DBOPEN(*dbName* as FileID, *fileName* as String, *NotUsed* as Integer)

Description
To open an existing file on the Device. Returns an integer result.

DbName is the reference to the file inside the program. All actions on the file will identify it by this name. *FileName* is the name of the file on the device. *NotUsed* should be 0. A FileID must be defined by "DIM *dbname* as Database" before being used in a DBOPEN command.

A DBOPEN is required before any subsequent actions on a file.

File Result Codes

-1	EOF on DbGet
0	Operation Successful
1	Operation Failed
2	Key not found - next higher key returned
3	File opened in read only mode
513	Memory Error
514	Index Out Of Range
515	Invalid Parameter
516	Read Only
517	File Open
518	Can't Open
519	Can't Find
520	Record In Wrong Card
521	Corrupt File
522	Record Deleted
523	Record Archived
524	Not Record DB
525	Not Resource DB
526	ROM Based or invalid File name
527	Record Busy
528	Resource Not Found
529	No Open File
530	Invalid Category
531	Not Valid Record

532	Write Out Of Bounds
533	Seek Failed
534	Already Open For Writes
535	Opened By Another Task
536	UniqueID Not Found
537	Already Exists
538	Invalid File Name
539	File Protected
540	File Not Protected

Example

```
Dim result as integer
Dim CustomerDb as database
Result=DbOpen(CustomerDb, "Company-Test", 0)
```

DbPosition Function

DBPOSITION(*dbName* as FileID, *dbRecNo* as Double, *dbOffset* as Integer)

Description

Positions the current file position to the record number as identified as *dbRecNo* and to the offset within the record as specified by *dbOffset*. The next DBGET or DBPUT will read or write data at this position. If *dbRecNo* is greater than the current number of records, the file is expanded to have *dbRecNo* records. The file must have been previously opened with DBOPEN. Returns an integer result. See DBOPEN for a table of result codes.

Example

```
Dim res as integer
Res=DbPosition(OrderDb, 15, 20) 'Position to rec 15, offset 20.
```

DbPut Function

DBPUT(*dbName* as FileID, *aVariable* as vartype)

Description

To write the value of *aVariable* to the file at the current position. The file must have been previously opened with DBOPEN and the position set using DBPOSITION. Returns an integer result. See DBOPEN for a table of result codes.

Example

```
Dim res as integer
Res=DbPut(OrderDb, OrderNo)
```

DbRead Function

DBREAD(*dbName* as FileID, *dbKey* as anyVarType, *varllist*)

Description
To read values from the record identified by the key value into the list of variables as specified by *varlist*. The file must have been previously opened with DBOPEN. Your *varlist* must match the datatypes of the fields exactly as they were written out. You cannot skip fields. All variables in the varlist must be variables: they cannot be expressions such as Field.text. Returns an integer result. If the exact key is not found, the next higher key will be returned and the result code will be 2. See DBOPEN for a table of result codes.

Example
```
Dim res as integer
Res=DbRead(CustomerDb, "674", custName, custAddress)
```

DbReadNext Function

DBREADNEXT(*dbName* as FileID, *dbKey* as anyVarType, *varlist*)

Description
Reads the next record from the file in key sequence and sets *dbKey* to the value of the key found. Reads data for all variables in *varlist*. The file must have been previously opened with DBOPEN. Returns an integer result. See DBOPEN for a table of result codes. For DBREADNEXT to work properly, the position must have been set with DBRESET, or a valid record found with DBREAD, DBREADNEXT or DBREADPREV.

Example
```
Dim res as integer
Dim theKey as String
Res=DbReadNext(CustomerDb, theKey, custName, custAddress, custPhone)
```

DbReadPrev Function

DBREADPREV(*dbName* as FileID, *dbKey* as anyVarType, *varlist*)

Description
Reads the previous record from the file in key sequence and sets *dbKey* to the value of the key found. Reads data for all variables in *varlist*. The file must have been previously opened with DBOPEN. Returns an integer result. See DBOPEN for a table of result codes. For DBREADPREV to work properly, the position must have been set with a valid record found with DBREAD, DBREADNEXT or DBREADPREV.

Example
```
Dim res as integer
Dim theKey as String
Res=DbReadPrev(CustomerDb, theKey, custName, custAddress, custPhone)
```

DbReset Function

DBRESET(*dbName* as FileID)

Description
Resets the file so DBREADNEXT will read the first record in the file. The file must have been previously opened with DBOPEN. Returns an integer result. See DBOPEN for a table of result codes.

Example
```
Dim res as integer
Res=DbReset(CustomerDb)
```

DbUpdate Function

DBUPDATE(*dbName* as FileID, *dbKey* as anyVarType, *varlist*)

Description
Updates an existing file record identified by *dbKey* and uses *varlist* to obtain new values to place into the file. The file must have been previously opened with DBOPEN. Returns an integer result. See DBOPEN for a table of result codes.

Example
```
Dim res as integer
Res=DbUpdate(CustomerDb, "654" , custName, custAddress, custPhone)
```

DegToRadians Function

DEGTORADIANS(*theAngleInDegrees* as float)

Description
Returns a float with an angle in degrees to radians. Most trig functions expect an angle in radians.

Example
```
x=DegToRadians(180)
y=cos(x)
```

Delay Statement

DELAY*sec*

Description
Puts the program into a wait state for *sec* seconds.

The *sec* variable may be a floating point variable with decimal fractions. In fact, a value for *sec* of under 1 second is allowable.(e.g. `DELAY 0.25` would delay a quarter second)

Example
```
Delay 1 'delays 1 second
Delay 2.5 'delays 2 and a half seconds
```

DestroyWindow Statement

DESTROYWINDOW *winName* as string

Description
Destroy the created graphics window *winName.* DestroyWindow commands must be done in the reverse order of the CreateWindow commands that were executed.

Example
```
DestroyWindow "barChartWin"
```

Dim Statement

DIM *varname* AS *type*
DIM *varArray*(*nnn*) AS *type*
DIM *varname* AS *type***length* [,*decimalPlaces*]
DIM *varArray*(*nnn*) AS *type***length*[,*decimalPlaces*]
DIM varName AS DATABASE dbName, dbRec, dbLayout [, key]

Description
Defines a variable to be used within a function or subroutine. If the variable is an array , then *nnn* defines the number of elements to be reserved.

Type	
Byte	a single character of data. On database input/output, a single character is written.(without the Terminating null character).
Database	used to define a file reference (FileID) that will be used in file commands. If the file is to be bound to a grid control, additional arguments are needed.
Date	used to store date values. File i/o results in a 64-bit floating point number. Dates are stored internally as (year-1900)*10000+month*100+ day
Double	Same in all respects to Float
Float	used to store numbers that may have both integer and fractional digits. On input/output to files, a 64-bit (8-byte) double precision floating point is used
Integer	used to store whole numbers (no decimal positions to the right of the decimal point). On input/output to files, a 32-bit(4-byte) integer is used.
Short	Similar to Integer except that file i/o results in a 16-bit(2-byte).
Single	Same as Float except file i/o results in a 32-bit(4-byte)single-precision floating point.
String	Maximum size is 32767 characters.
Time	used to store time values. File i/o results in a 64-bit floating point number. Times are stored internally as hour*10000+minute*100+seconds
UserType	see the TYPE and END TYPE statements
Variant	see SysTrapFunc and SysTrapSub functions

Length defines how many digits get displayed. *DecimalPlaces* is the number of digits to the right of the decimal point. This must be at least 2 less than *length.*

If we are declaring a file for use in a grid. VarName is the name of the variable. *DbName* is the name of the file on the Device. *DbRec* is a global variable which will be created to contain the contents of the current record. It should not be dimensioned already. *DbLayout* is the layout of the record: it should already be defined using a TYPE statement. *Key* is an optional variable that must already be dimensioned: it is the key that is used for the *dbName* file. This form is only used to define *Varname* when used as the first argument of a BindToDataBase method for a Grid control.

Example

```
Dim income as float*12,2
Dim count as integer
Dim yesterday as date
Dim appt as time
Dim rates(12) as float
Dim Orders as database
Type dbBluesLayout
    Name as String
    age as Integer
    active as Integer
End Type
Dim blueKey as Integer
Dim dbBlues1 as Database "Blues" Creator "Grid" Keyed With Record BlueKey,
dbBluesRec as dbBluesLayout
```

Display Statement

DISPLAY *varList*

Description
Displays the list of variables at the current screen cursor location.

Example

```
Cursor 10,10
Display custLastName, ",", custFirstName
Cursor 10,20
Display custAddress
```

Do / Loop Statement

DO
DO UNTIL *expression*
DO WHILE *expression*

Description
Repeats execution of the statements between the DO statement and the LOOP statement until an EXIT DO statement is executed, the *expression* in the DO WHILE is false or the *expression* in the DO UNTIL is true.

Example

```
i=0
Do
  i=i +1
    if i=5 then
    Exit Do
  End if
  ..............
  .............
Loop

'An equivalent  is
i=0
Do Until i=5
  i=i+1
  ...........
  ...........
Loop

'An equivalent  is
  i=0
  Do While i < 5
  i=i+1
  ............
  .............
Loop
```

DrawBitmap Statement

DRAWBITMAP *bitmapId* as integer, *xStart* as Integer, *yStart* as Integer

Description
To display a bitmap image on the graphics screen starting at the starting *xStart* and *yStart* location. A bitmap file with the given *bitmapId* must be included in the project.

Example
```
DrawBitmap 1027, 10, 25
```

DrawChars Statement

DRAWCHARS *theChars* as String, *xStart* as Integer, *yStart* as Integer[,*penType*]

Description
To display characters on the graphics screen starting at the *xStart* and *yStart* location. *PenType* is optional—if specified, it must be nsbNormal, nsbInverted, or nsbGrey.

Example
```
DrawChars "The starting salary is $500", 10, 25
DrawChars "Rate",30,10,nsbInverted
DrawChars Str(CommissionRate),40,10,nsbGrey
```

DrawLine Statement

DRAWLINE *xStart* as Integer, *yStart* as Integer, *xEnd* as Integer, *yEnd* as Integer[, *penType*]

Description
To draw a line on the graphics window between the starting and ending coordinates. *PenType* is optional—if specified, it must be nsbNormal, nsbInverted, or nsbGray.

Example
```
DrawLine 10,10, 20,40
DrawLine 20, 40, 20, 60, nsbGray
```

DrawRectangle Statement

DRAWRECTANGLE *xStart* as Integer, *yStart* as Integer, *width* as Integer, *height* as Integer, *cornerDiam* as Integer [, *penType*]

Description
Draws a rectangle border using the specified coordinate, *width*, and *height*.
CornerDiam is the corner radius which can be used to create round corners (a corner diameter of zero causes square rectangles to be drawn). *PenType* is optional—if specified, it must be nsbNormal, nsbInverted, or nsbGray.

Example
```
DrawRectangle 10,10,30,40,0
DrawRectangle 40,50, 10,15,3,nsbInverted
```

End Function Statement

END FUNCTION

Description
To mark the end of a function definition

Example

```
Function myFunc
  .......................
  ......................
End Function
```

End If Statement

END IF

Description
To mark the end of an If statement. The single word ENDIF is also allowed.

Example

```
If a=1 then
  .........................
  ........................
End if
```

End Sub Statement

END SUB

Description
To mark the end of a subroutine definition

Example

```
Sub mySub
  ..........
  ...........
End Sub
```

End Type Statement

END TYPE

Description
To mark the end of a user-defined type definition

Example

```
Type userType
  x as double
  y as string
  .........
  ...........
End Type
```

EraseWindow Statement

ERASEWINDOW

Description
Erases the current graphics window. See CreateWindow for more information.

Example
```
EraseWindow
```

Exit Do Statement

EXIT DO

Description
To exit from a DO loop and continue execution at the statement after the LOOP statement.

Example
```
Do
  If i=5 then
    Exit Do
  End if
  ....................
  ....................
Loop
```

Exit For Statement

EXIT FOR

Description
To exit from a FOR...NEXT loop and continue execution at the statement after the Next statement

Example
```
For i=1 to 20
  If mileage(i) > 5000 then
    Exit For
  End if
  ....................
  ....................
Next
```

Exit Function Statement

EXIT FUNCTION

Description
To exit a function and resume execution in the calling program

Example
```
Function a(count as integer) as Float
  If count > 100 then
    A=0
    Exit Function
  End if
  A=Count * 1.5
End Function
```

Exit Sub Statement

EXIT SUB

Description
To exit a subroutine and return control to the calling program.

Example
```
Sub mySub(count as Integer , result as integer)
  If count > 100 then
    Result=0
    Exit Sub
  End if
  Result=count * 1.5
End Sub
```

Exp Function

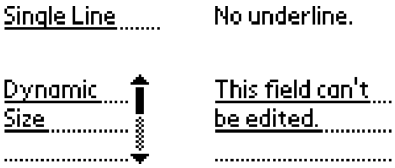
EXP(*theNumber* as Float)

Description
Returns a float with the exponential of a number (e raised to the number power).

Example

```
Dim result as Float
Result=Exp(16.0) 'result would be "e" raised to the 16th power
```

Field Object



A FIELD object is an area on the form where the user can key text. A value can also be set by the program. If you want to force a line feed in the text, use the &h0A character (chr(10)).

The code for a Field is executed when the user exits the field by tapping on some other object or menu item. The code may then perform editing for validity or other computations dependent on the new field value. This code will be triggered even if the user does not change the field value (e.g. taps on the field but then makes no changes) and positions on a different field. NS Basic/Symbian OS does not keep track of the before and after field values; it only knows that the field was selected for editing and then the user moved off the field to another object.

A common program need is to read a record from a file and display the contents of fields to the user, let the user make changes, and then restore the new values to the record. This would normally mean that in the form startup code, you would read the record and place values into each field.

Properties Supported (Set at design time)
Label, Font ID, Text, Max Characters, Left Justified, Underline, Single Line, AutoShift, HasScrollbar, Dynamic Size, Editable, Numeric, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Redraw, Text, SetFocus, ID, Index, Type

Example
Read in record and display it. Let the user make changes and write it out.

```
Dim Name as string
Dim Age as integer
Dim Married as integer '0=not or 1=married
Dim err as integer
Err=DbRead(MyDb, Name, Age, Married) 'using Name as key
FldName.text=Name
FldAge.text=str(Age)
ChkMarried.status=Married
```

Then, on the "Done" button that the user presses after changing the values on the form, you might have the code:

```
Dim Name as string
Dim Age as integer
Dim Married as integer
Dim err as Integer
If not testNum(fldAge.Age, " ",3,0)=0 then
  Beep
  Msgbox "Age field is not valid."
  Exit sub
end if
Name=fldname.text
Age=val(fldAge.text)
Married=chkMarried.status
Err=DbUpdate(MyDb, Name, Age,Married)
```

FillRectangle Statement

FILLRECTANGLE *x* as integer, *y* as integer, *width* as integer, *height* as integer,
cornerDiam as Integer [, *penType*]

Description

Fills a rectangle border using the specified *x* and *y* coordinates, *width*, and *height*. *CornerDiam* is the corner diameter which can be used to create round corners A corner diameter of zero causes square rectangles to be drawn. *PenType* is optional—if specified, it must be `nsbNormal`, `nsbInverted`, `nsbGray`, or nsbCustom, For more information on nsbCustom, see the Pattern sample.

Example

```
FillRectangle 10, 15, 30, 40, 5, nsbGray
```

FirstOfMonth Function

FIRSTOFMONTH(*theDate* as Date)

Description

Returns a date which is the 1st of the month from theDate

Example

```
Dim theDate as Date
Dim result as Date
theDate=DateVal(1999,2,15)
result=FirstOfMonth(theDate) 'will calculate result a date of 2/1/1999
```

Floor Function

FLOOR(*theNumber* as Float)

Description

Return an number by rounding *theNumber* to the next lower integer. Negative numbers are rounded to the next lower negative number.

Example

```
Dim x as float
Dim result as integer
x=-33.2
result=floor(x) 'will calculate result=-34
```

For Statement

FOR *varName*=*startValue* TO *endValue* [STEP *stepValue*]

Description
Defines the beginning of a repeated set of statements. The end of the repeated set of statements is defined by the NEXT statement. *varName* is initially set to the *startValue* and incremented by the optional *stepValue* on each repetition of the sequence. If *stepValue* is not specified, then a *stepValue* of 1 is used. The execution of the set of repeated statements ends when the value of *varName* is greater than *endValue*. At that point, the next statement executed is the statement following the NEXT statement. In the set of repeated statements an EXIT FOR statement immediately cancels the execution of the set of repeated statements and transfers control to the statement following the NEXT statement. *StepVal* may be a positive or negative value.

Example
```
For i=1 to 5
  .........
Next

For cnt=cntMax to 1 step -1
  .........
  if total > 5000 then
    Exit For
  End if
  .........
next
```

Form Object

A FORM is a special case of object. It acts as a container for other objects. Only only one Form can be active at a time. Use NextForm to move from one Form to another, When a Form is displayed, its From Before and Form After scripts are executed. If an event occurs on a form, it is passed to the form's Event Code.

Properties Supported (Set at design time)
Name, ID, Default Form, Height, Left, Modal Form, Show TitleBar, Title, Top, Width

Methods Supported (See "Methods")
Clear, Count

Example
```
Form1003.clear
Msgbox "The number of objects on this form is" + str(Form1003.count)
```

Format

FORMAT(*numVar* as anyNumType, *fmtstring* as String)

Returns: A formatted text string

Description
To format numeric data into text strings suitable for display or printing. *NumVar* is any numeric format including integer, short, float, or double. *Fmtstring* is a string which controls the formatting of the numeric data. The variable being assigned to should be padded with enough characters to receive the result. *fmtstring* may contain one or more of the following characters:

0	replaced by a digit from the number if available or zero if not
n	replaced by a digit from the number if available or space if not
#	replaced by a digit from the number if available or nothing if not
.	places a decimal point in the output text
+	Replaced by a minus sign if the number is negative or space if positive
-	Replaced by a minus sign if the number is negative or space if positive
,	places a comma in the output string if a leading digit has been encountered or a space if not.
	Any other character is copied into the output stream

Example

```
Dim a as double
Dim b as double
Dim c as double
Dim i as integer
Dim j as integer
Dim k as Integer
Dim s as String
a=123.45
b=-37.65
c= 12345.6
i=92
s="        "
s=Format(a, "#####.000" ) 'result= 123.450  (length=7 because # does not reserve a
blank if there is no leading digit)
s=Format(a, "nnnnn.000" ) 'result=123.450(length=9 because n reserves a blank if
there is no leading digit)

s=Format(a, "-00000.000") 'result=  00123.450 (minus is ignored because the number
is negative)
s=Format(b,"+nn,nnn.0") 'result is -37.6
s=Format(c, "nn,nnn.00") 'result is 12,345.60
s=Format(i, "###%")    'result is 92%
```

Function Statement

FUNCTION *functionName*(*varList*) AS *resType*

Description
Defines the start of a function, the variables that are passed as arguments and the type of result returned. A function always returns a result to the calling program. Each variable in *varList* is defined as *Varname* AS *varType* where *varType* is the type of variable(Integer, Float, etc). Variables are separated by commas. This can be any of the valid data types(Integer, Float, array, UDT etc.), except for the file type.

A FUNCTION statement defines the start of a function and the end of the function is defined by the END FUNCTION statement. A function's return value is set somewhere in the function's executable statements by assigning the function's name to a value or computation. Array and UDT datatypes may not be returned.

When END FUNCTION is executed, the function returns control to the calling program. Parenthesis ("()") are required if there are no arguments.

Example

```
Function Calc1(rate as Float, loanAmt as Float) as Float
  Calc1=rate * loanAmt 'this value is returned
End Function
```

Gadget Object

A GADGET item is a rectangular area on the form that acts like a button where you are responsible for drawing on or writing text on the area. A gadget item has no displayed image or text. When the user taps on the object, the program code associated with the object is executed.

Gadgets can also be used to capture signatures and drawings as bitmaps. See Section 10.I and the sample for more information on using Signatures.

Properties Supported (Set at design time)
Left, Top, Width, Height, Visible

Methods Supported (See "Methods")
Hide, Show, Redraw, StartSignatureCapture, EraseSignature, DisplaySignature, EndSignatureCapture, ID, Index, Type

GetEventType Function

GETEVENTTYPE()

Description
Returns the event number which triggered special event processing. This function will also return event codes sent from Libraries. The return values are

	Description	Greater than 24832
NsbFormClose	Current form closed	9
NsbFormUpdate	Current form updated by OS (not by user)	8
NsbKeyOrButton	key or button pressed	1
Nsb5Way	Five way button key code	7
NsbPenDown	PenDown event (stylus pressed on the screen)	2
NsbPenUp	PenUp event (stylus lifted from the screen)	3
NsbSerialEvent	Serial data received	6
NSBTimer	A timer has expired. See SysInfoSet.	10

Example
Ans=GetEventType()

GetKey Function

GETKEY()

Returns: a string of one character length which is the last key or button pressed. Certain events also produce GetKey sequences.

Description
Returns the last key or button pressed as a one-character string value. See the example below for the values which are returned. See the SpecEvent.prj sample in the Samples folder. SpecEvent also shows how to recognize the 5 Way buttonTo see the raw characters codes that come in from events, you can also use the SysInfo(9), SysInfo(10) and SysInfo(11) calls.

Example
In the Form Event Code of a form, use this:

```
dim key as string
If Not GetEventType()=NSBKeyOrButton then Exit Sub
key=getKey()
select case asc(key)
case 1    'hard button #1 on bottom of case
...
case 2    'hard button #2 on bottom of case
...
case 3    'hard button #3 on bottom of case
...
case 4    'hard button #4 on bottom of case
...
case 5    'menu key on silkscreen area
...
case 17    'application launch key on silkscreen area
...
case 7     'find key on silkscreen area
...
case 16   'calculator key on silkscreen area
...
case 11   'up button
...
case 12   'down button
...
case 14    'power on/off button
...
case 15    'cradle hotsync button
...
case 18    'auto power off
...
case else
...
end select
```

GetPen Function

GETPEN *currentXPos* , *currentYPos* , *penStatus*

Description
Returns the last clicked onto or removed from the screen in the *currentXPos* and *CurrentYPos* variables. Returns the pen status in the *PenStatus* variable as NsbPenUp or nsbPenDown.

Example
```
GetPen lastX, lastY, upDownStatus
```

GetVersion Function

GETVERSION (*PRCName*)

Description
Returns the version string (tver) of a program. *PRCName* is the name of the program. On the Device, program names do not normally have prc extensions.

Example
```
Dim s as string
S=GetVersion("NSBRuntime")    'result is 4.0.0
```

Global Statement

Global *varname* as type
Global *varArray*(*nnn*) as type
Global *varname* as type**length*[,*decimalPlaces*]
Global *varArray*(*nnn*) as *type***length*[,*decimalPlaces*]

Description
Defines a variable that has a global definition; that is, it can be used in any code module in the entire program. This is the different than variables defined in a DIM statement that are only usable in the function or subroutine in which the DIM statement occurs. For more on DIM, see the DIM statement. The GLOBAL statement is normally used in your Startup Code.

Example
```
Global amount as Float
Global warehouseNames(25) as String
```

Gosub Statement

GOSUB *labelName*

Description
To transfer to another point (defined by the *labelName*) in the current Sub or Function. The code transferred to may return to the statement following the GOSUB statement by executing a RETURN statement..

Example

```
Gosub routine1
...
routine1:
  ....
  ....
Return
```

Goto Statement

GOTO *labelName*

Description
To transfer control within the current Sub or Function to the label statement which has the label of *labelName*. Don't expect much respect from other programmers if you use this frequently.

Example

```
If count=5 then
  GoTo NoMoreCalculations
End if
.........
NoMoreCalculations:
......
```

Grid Object

The Grid object allows you display a table of data, supplied either from your program or automatically loaded from a file. You define the basic appearance of the grid at design time and populate it with data at runtime.

When a cell in the grid is tapped, the grid's row and col properties are set and can be used in your program. While the fields in the grid cannot be directly edited by the user, you can change their values using TextMatrix and ValueMatrix for string and numeric fields respectively. Text and Value can be used for the currently selected cell.

Please see Section 13 of the Handbook for a more detailed description of the use of this object.

Properties and Methods Specific to this Object

.BINDTODATABASE *dbName*, *dbFieldNameList* [Where *condition*]	Automatically loads a grid with data from a file. *DbName* is a file variable, set up by a previous DIM WITH RECORD statement. *DbFIeldNameList* is a list of fieldnames in the file, or the name of a Type structure in the file. *Condition* is an optional argument which selects which records to display. The format is the same as in an IF statement.
.COL	Get or set the current column. Range is 1 to COLS.
.COLS	Get the number of columns
.COLTYPE *colNo*, *type*, *formatString*	Set the type and format of a column. Columns default to text with null values. If you want a different type of data in a column, you must use this method to set it at runtime. *ColNo* is an existing column number in the Grid. *Type* is either "text","numeric" or "checkbox". *FormatString* depends on *type*: for "text", it will be the default value of the cell. For "numeric", it is a format string as used in the FORMAT statement. For "checkbox", the *formatString* is displayed to the right of a checkbox
.COLWIDTH(*colNo*)	Get or set the width of column *columnNo*. To hide a column, set to -1.
.FONTMATRIX(*rowNo*, *ColNo*)	Get or set the Font Number (0-7) for the cell.
.HIDEGRIDLINES	Hide the lines of the grid.
.ROW	Get or set the current row. Range is 1 to ROWS. Set this to 0 to unselect grid.
.ROWDATA	Get or set the rowdata value of the row. This is a user defined value.
.ROWS	Get or set the number of rows.
.SHOWGRIDLINES	Show the lines of the grid.
.TEXT	Get or set the value of the current cell with text in it.
.TEXTMATRIX(*rowNo*, *ColNo*)	Get or set the text value of the cell at *rowNo*, *ColNo*.
.TOPROW	Get or set the top row to display
.VALUE	Get or set the value of the current cell with a number in it.
.VALUEMATRIX(*rowNo*, *ColNo*)	Get or set the numeric value of the cell at *rowNo*, *ColNo*

Properties Supported (Set at design time)
Cols, Has Scrollbar, Height, Left, Top, Visible Rows, Width

Other Methods Supported (See "Methods")
Add, Clear, Hide, HideGridLines, Redraw, Remove, Show, ShowGridLines

Hour Function

HOUR(*theTime* as Time)

Description
Returns an integer with the hour value from a time.

Example
```
Dim theTime as Time
Dim result as Integer
theTime=TimeVal(13,24,15)
result=Hour(theTime) 'will calculate result=13
```

HourMin Function

HOURMIN(*theTime* as Time)

Description
Returns a string with the hours and minutes from *theTime* in HH:MM format.

Example
```
Dim theTime as Time
Dim result as String
theTime=TimeVal(13,24,15)
result=HourMin(theTime) 'will calculate result="13:24"
```

HourMinAMPM Function

HOURMINAMPM(*theTime* as Time)

Description
Returns a string with hours, minutes, and AM or PM indicator in the format "HH:MM XX" where XX="AM" or "PM"

Example
```
Dim theTime as Time
Dim result as String
theTime=TimeVal(13,24,15)
result=HourMinAmPm(theTime) 'will calculate result="01:24 PM"
```

If /Then/ Else / End if / ElseIf Statement

```
IF condition THEN
   Statements group 1
END IF
or
IF condition THEN
   Statements group 1
ELSE
   Statements group 2
END IF
or
IF condition THEN statement1
or
IF condition THEN statement1 ELSE Statement2
or
IF condition1 THEN
   Statement Group 1
ELSEIF condition2 THEN
   Statement Group 2
END IF
```

Description
To test a condition and then based on whether the condition is true of false, to conditionally execute other statements. If condition is true then the statements in Statement Group 1 are executed, otherwise the statements in Statement Group 2 are executed.

Condition may be a complex logical condition including the following elements:
`AND` operator
`OR` operator
`NOT` operator
Parentheses
>(Greater than sign)
<(Less than sign)
=(Equal sign)
<=(Less than or equal sign)
>=(Greater than or equal sign
<>(not equal sign)

Example

```
If amt=1000 then
  Goto allDone
End if

If state="IL" then
  Gosub IllinoisCalcs
Else
  Gosub OtherSatesCalcs
End if

If state="IL" Or state="CA" then
  Goto noTaxes
End if

If not salary <=2000 then 'This demonstrates the NOT operator
  Taxes=salary * .32
End if
```

```
If CountyCode=20 then 'this demonstrates nesting of IF statements
  TaxRate=.043
Else
  If CountyCode=21 then
    TaxRate=.041
  Else
    TaxRate=.039
  End if
End if

If(Sex="Male" and Weight > 200) or (Sex="Female" and weight > 180) then
  Msgbox "Lose Weight!"
End if

If Amount >10000 Then
  LoanProcess1
Elseif Amount > 5000 Then
  LoanProcess2
Elseif Amount > 1000 Then
  LoanProcess3
End if

If Sex="Male" Then AgeLimit=65 else AgeLimit=50
```

InStr Function

INSTR(*start* as integer, *s* as string, *pattern* as string, *type* as integer)

Description
Searches string *s* starting at position start to find the string pattern.If *type* is 0 then a search for an exact match of case and pattern occurs. If *type* is 1, then a case insensitive search is conducted. It returns an integer result which is 0 if *pattern* is not found in string *s* or an integer which is the position within string *s* where the *pattern* was found.

Example
```
Dim s as string
Dim result as integer
s="abcdef"
result=InStr(1, s, "cde", 0) 'would return result=3
result=InStr(1, s, "CDE", 0) 'would return 0 because of case difference
result=InStr(1, s, "CDE", 1) 'would return 3 because case is ignored
```

Int Function

INT(*theNumber* as Float)

Description
Returns the integer portion of a number.

Example
```
Dim x as float
Dim result as integer
x=-33.2
result=Int(x) 'will calculate result=-33
```

Label Object

LABEL objects are used to display text on the form. Label objects are often used for Titles, instructions, or labels preceding other field controls. The Text of a label can be modified by the program, but it cannot be made longer than it was at design time. Use blanks to fill it out if necessary.

Properties Supported (Set at design time)
Left, Top, Font ID, Label

Methods Supported (See "Methods")
Hide, Show, Redraw, text, ID, Index, Type

Example
```
MyLabel.text="Enter Name:"
MyLabel.redraw
```

Label Statement

labelName:

Description
Define a LABEL within a sub or function that may be transferred to with a GOTO or GOSUB statement. The label must be the only thing on the statement line.

Example
```
if loanType="home" then
  GoTo homeLoanCalculations
Else
  GoTo carLoanCalculations
End if
.........
homeLoanCalculations:
.........
GoTo moreCode
CarLoanCalculations:
........
moreCode:
........
```

LastOfMonth Function

LASTOFMONTH(*theDate*)

Description
Returns the date which is the last day of the month from *theDate*

Example
```
Dim theDate as Date
Dim result as Date
theDate=DateVal(1999,6,15)
result=LastOfMonth(theDate) 'will calculate result a date of 6/30/1999
```

LCase Function

LCASE(theString)

Description
Returns a string result that is the lower case conversion of the input string

Example
```
Dim result as String
Result=Lcase("John Doe") 'would set result to "john doe"
```

Left Function

LEFT(*theString* as String, *num* as Integer)

Description
Returns the leftmost *num* characters of *theString* . If *theString* has less than *num* characters, then the result will be *theString*.

Example

```
Dim result as String
Result=Left("John Doe", 6) 'would set result to "John D"
```

LeftPad Function

LEFTPAD(*theString* as String, *num* as Integer)

Description
Returns a string that is *num* characters long. If the input string, *theString*, is less than *num* characters long, then spaces are inserted before the value of *TheString* to pad on the left.

Example

```
Dim result as String
Result=LeftPad("John Doe", 12) 'would set result to "    John Doe"
```

Len Function

LEN(*theString* as string)

Description
Returns the integer length in characters of the input string.

Example

```
Dim result as Integer
Result=Len("John Doe") 'would set result to 8
```

Let Statement

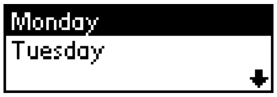
LET *varName=expression*

Description
Assigns the results of an arithmetic or string expression to the variable which occurs on the left side of the equal sign. *expression* may be any combinformation of arithmetic operators, variable names, array names, functions and parentheses.

LET is assumed if you supply a statement of the :
Varname=expression
The following 2 statements are equivalent:
Let x=2 * y
X=2 * y

Example
```
Let salary=commisionRate * unitsSold
Let monthlySales(i)=monthlyUnits(i) * averageSalesPrice
Let x=sqrt(Abs(distance) / milesPerHour) ^ 2)
name=Proper(name)
```

List Object



A LIST object is a box on the form where multiple text strings (items) can be displayed with each text string (item) on a separate line. One item of the list may be highlighted (selected) and is printed in reverse colors with white letters on a black background. The box area is only big enough to display a few items. If there are more items than there is room to display, there is a scrollbar along the right side of the list box to allow up/down scrolling. The Selected method is set before the list code is run, while the Text is set afterwards.

List contents may be set at design time by filling in the list box that appears by clicking on the List property. List contents may also be set at execution time using the Add method. The best place to do this is in the form's After Code for the form on which the list object appears. Use the NODISPLAY option to keep the form from refreshing each time you add an item. Set .Selected to 0 to not select any item.

Properties Supported (Set at design time)
Font ID, Visible Items, Visible, List

Methods Supported (See "Methods")
Left, Top, Width, Hide, Show, Redraw, Add, Remove, Clear, Text, Selected, NoItems, ID, Index, Type

Example
```
Dim ItemsOnList as Integer
Dim selectedname as String
ListCity.clear 'always clear the list first
ListCity.add "New York",NODISPLAY     'delay form update
ListCity.add "Boston",NODISPLAY       'delay form update
ListCity.add "New Orleans"            'now, update form
ListCity.selected=2 ' Highlight Boston
ListCity.remove 3 'Delete New Orleans
ItemsOnList=ListCity.NoItems 'get items on list
selectedName= ListCity.text(ListCity.selected)
ListCity.clear
```

LoadLibrary Statement

LOADLIBRARY *filename as string [,refname as string]*

Description
LoadLibrary creates a code object that can then be used to access external procedures found in the related Library. It opens Library *filename* as *filename* or *refname* if supplied. See Tech Note 5 for complete information on using Libraries.

Example
```
Dim username as String
LoadLibrary "NSBSystemLib",”NSL”      'Uses NSBSystemlib.inf, refname = NSL
userName = NSL.SyncUserName()          'Gets Hotsync ID
```

Log Function

LOG(*theNumber* as Float)

Description
Returns a float with the natural log of a number.

Example
```
Dim result as Float
Result=Log(16.0) 'result would be the log of 16
```

Log10 Function

LOG10(*theNumber* as Float)

Description
Returns a float with the base-10 log of a number

Example
```
Dim result as Float
Result=Log10(16.0) 'result would be the base-10 log of 16
```

LTrim Function

LTRIM(*theString* as String) As String

Description
Returns a string with all leading space characters from the start of the string removed.

Example
```
Dim result as String
Result=Ltrim(CityName)
```

MenuDraw Statement

MENUDRAW *menuName* as string

Description
To draw the specified menu across the top of the display screen. Menus are defined using the menu editor.

Example

```
If not GetEventType()=NSBKeyorButton then Exit Sub
If asc(getkey())=5 then
  Menudraw "main"
  SetEventHandled
End if
```

MenuErase Statement

MENUERASE

Description
Erases the menu drawn across the top of the display screen. In most cases, you won't have to use this statement. Selecting a menu item or tapping outside the displayed menu will cause the menu to erase.

Example
MenuErase

MenuReset Statement

MENURESET

Description
Causes any dropdown submenu to be cancelled. The top level menu will still remain drawn across the top of the display screen.

Example

```
MenuReset
```

Methods **Object**

Methods change the appearance or settings of an object. Each object has certain methods that can be used. Check the documentation for the Object to find which ones of these methods applies to it.

Add *Itemtext*, *indexNo*[,NODISPLAY]	Add *ItemText* to list at *indexNo*. NODISPLAY delays updating List Objects until complete. For Grid objects, there are no arguments: a new row is added to the bottom of the table.
Clear	Clear a listthe object. If the object is a form, all fields are set to empty. (""). If it is a Grid, all rows are deleted.
Count	Returns the number of objects for a Form object.
Hide	Hide the object
ID	Get the object's internal ID number (integer), Useful for API calls.
Index	Object's position on form. First position is 0. If the form has a title, it is Index 0.
ItemNo *item[, index]*	Add *item* to list at position *indexNo*. If *indexNo* is 1, add to beginning of list. If no *indexNo* supplied, add *item* to end of list.
Itemtext *IndexNo*	Get item in a list
Max *int*	Get or set maximum value of scrollbar
MaxChars *int*	Get or set maximum characters
Min *int*	Get or set minimum value of scrollbar
NoItems	Get number of items in a list
PageSize *int*	Get or set page size value of scrollbar
Redraw	Refresh the object
Remove *indexNo*	Remove row *indexNo* from listobject
Selected *indexNo*	Get or set selection
SetFocus	Set focus to object
Show	Show the object
Status	Used by Checkbox and Pushbutton
Text	Get or set the text or label
Value *CurValue*	Get or set current value of scrollbar

Example

```
MyField.text="Reg Llama" 'set text of a field
MyPopupList.add "Brixton" 'add a name to a popup list
MyCheckbox.setFocus 'set the focus to a field
Msgbox MyOtherField.text 'output the value of a field
MyCheckbox.status=nsbChecked 'set checkbox
```

Mid Function

MID(*theString* as String, *start* as Integer, *num* as Integer)

Description
Returns a substring from *theString* starting at *start* and continuing for *num* characters. If *start* is beyond the length of *theString* then a null string results. If *start* + *num* is exceeds the length of *theString*, then the result will be the rightmost portion of *theString*, starting with the character at position *start*.

Example

```
Dim result as String
Result=Mid("John Doe", 3, 4) 'would set result to "hn D"
```

Minute Function

MINUTE(*theTime* as Time)

Description
Returns an integer with the minute value from a time

Example

```
Dim theTime as Time
Dim result as Integer
theTime=TimeVal(13,24,15)
result=Minute(theTime) 'will calculate result=24
```

MMDDYYToDate Function

MMDDYYTODATE(*theString* as String)

Description
Converts *theString* from the format "MM/DD/YY" or "MM/DD/YYYY" to an internal date result and returns it.

Example

```
Dim result as Date
Result=MMDDYYToDate("06/13/98") 'would set result to 1998/06/13
```

Mod Function

MOD(*theNumber* as Integer, *theDivisor* as Integer)

Description
Returns the remainder of dividing *theNumber* by *theDivisor.*

Example

```
Dim result as integer
Result=Mod(74 , 8) 'result will be 2
```

Month Function

MONTH(*theDate* as Date)

Description
Returns an integer with the months value from a date

Example
```
Dim theDate as Date
Dim result as Integer
theDate=DateVal(1995,6,23)
result=Month(theDate) 'will calculate result=6
```

MonthDay Function

MONTHDAY(*theDate*)

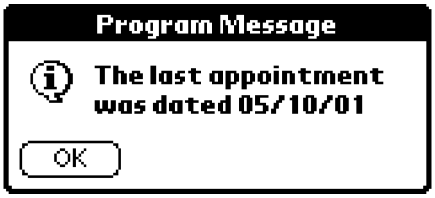
Description
Returns a string of the MM/DD from an internal Date type .

Example
```
Dim theDate as Date
Dim result as String
TheDate=DateVal(1995,6,23)
Result=Monthday(theDate) 'will give the string "06/23"
```

MsgBox Statement

MSGBOX *msg* as String

Description
Displays a *msg* within a box on the display screen and waits for the user to respond before continuing program execution. The maximum size of *string* is 185 characters. Use chr(10) to insert a new line.



The last appointment was dated 05/10/01

Example
```
MsgBox "The last appointment was dated " + lastDate
```

Next Statement

NEXT

Description
To end a FOR statement, which is a set of statements that are repeated a number of times based on the FOR parameters. The NEXT statement causes the FOR control variable to be incremented by the step increment and tested to see if the repeated statements should be executed again or termination should occur. When termination occurs, the statement following the NEXT statement is executed. FOR/NEXT loops may be nested to any level.

Example
```
Dim weight(5, 10 , 3)
For i=1 to 5
  For j=1 to 10
    For k=1 to 3
      Total=total + weight(i, j, k)
    Next
  Next
Next
```

NextForm Statement

NEXTFORM *formName* as string[, CLEAR]

Description
Hide the current form and transfer to another where *formName* is a string variable containing the name of the form to transfer to. If CLEAR is added to the end of the statement, the form is restored to its design time state, eliminating all changes made to the form including field values, checkbox and choicebox settings, emptying lists and popups, and eliminating any changes to object labels or text. If CLEAR is not specified, the form will retain its values if you return to it using NEXTFORM. The NEXTFORM statement should be the last statement you execute on the current form. Use Global variables to save variables that are shared by more than one form.

Example
```
Dim form as string
form="ShowMortgageRates"
NextForm form
```

NoOccurs Function

NOOCCURS(*theVariable theSubscriptLevel* as Integer)

Description
Returns an integer with the maximum value of a subscript for the given variable and subscript level(1 to 3)

Example
```
Dim x(10, 200, 5) as Float
Dim result as integer
result=NoOccurs(x, 1) 'result would be 10
result=NoOccurs(x, 2) 'result would be 200
result=NoOccurs(x, 3) 'result would be 5
```

Now Function

NOW()

Description
Returns the current time-of-day as recorded in the computer's Internal clock.

Example
```
Dim result as Time
result=Now()
```

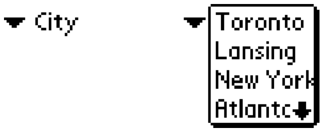
PlaySound Statement

PlaySound *Resource#*, *ampScale*, *Flags*

Description
The PlaySound statement allows you to play short pre recorded sounds in your program. It can only be used on Symbian OS 5.0 and later. *Resource#* is the resource number in your project. The sound must be in wav format and be less that 64k. Test the sounds on the device you want to use carefully: 16 bit mono, 8khz PCM encoded sound seems to work well. Add sounds to your project by doing "Add Resource", then setting the Type of the resource to 'wave'. *AmpScale* is the volume. The setting is from 0 to 32767. *Flags* determine how the sound plays. 0 means do not continue until the sound stops playing; 1 means to keep processing while the sound plays. See the sample SoundTest.prj in the Samples folder.

Example
```
PlaySound 1004, 32767, 1
```

PopUp Object



A POPUP object displays a text label with a graphic element [down-triangle] on its left. When tapped, multiple text strings (items) are displayed with each text string (item) on a separate line. One item in the list may be highlighted (selected) and will be printed in reverse video (white letters on a black background). The box size is set at compilation-time (cannot be changed during program execution), and when not large enough to display all items will contain page-up/page-down controls. List contents may be entered at design time in the List property, or at execution time using the .Add method. The best place to do this is in the form's "After" code. The Selected method is used to identify the selected item in the list. The Text method is used to set the text. The text may differ from the selected item until the Redraw method is used.

Properties Supported (Set at design time)
Label, Font ID, Anchor Left, Visible, List

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Redraw, Add, Remove, Clear, Text, Selected, NoItems, ItemText, ItemNo, ID, Index, Type

Example

```
Dim ItemsInList as Integer
Dim selectedName as String

'clear the list and populate it - do in Form After code
```

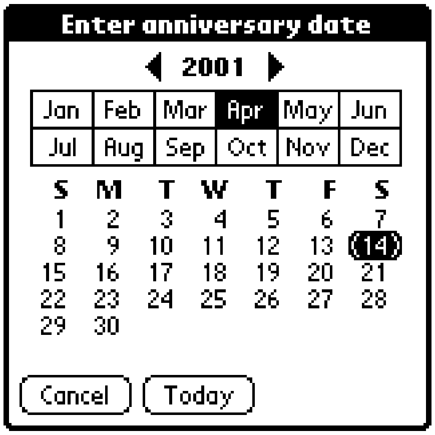
```
PopListCity.Clear

PopListCity.Add "New York" 'Populate the list
PopListCity.Add "Boston"
PopListCity.add "New Orleans"
PopListCity.Selected=2                'Select item#2 (Boston)
PopListCity.Text="What City?"                'Set text
PopListCity.Redraw                           'Show selected item in text
PopListCity.Remove 3                         'Delete item#3 (New Orleans)
ItemsInList=PopListCity.NoItems              'Get number items in list
selectedName=PopListCity.ItemText(PopListCity.Selected) ' get text of currently
selected item.
```

PopupDate Function

POPUPDATE(*DateVal* as Date, *title* as String)

Description
Shows the user a date and let the user change the date using the Device's standard popup for date input. A valid value for *DateVal* must be supplied when calling. If the user changes the date then the new date value will be placed in *DateVal* and the function will return a value of 1. If the user does not modify the date, the *DateVal* will not change and the function will return a value of 0. The *title* is displayed in the date input popup and can be used to prompt the user for the type of date desired. This function requires Symbian OS 3.3 or later. On some devices, calling this function will cause the After Script to be run on return.


Enter anniversary date

2001

Jan	Feb	Mar	Apr	May	Jun
Jul	Aug	Sep	Oct	Nov	Dec

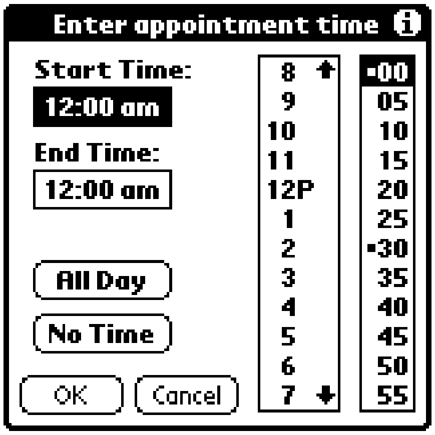
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Example
```
Dim theDate as date
Dim Res as integer
TheDate=today()
Res=PopupDate(theDate, "Enter anniversary
date")
```

PopupTime Function

POPUPTIME(*startTime* as Time, *endTime* as Time, *title* as string)

Description
Shows the user starting and ending times and let the user change these times using the Device's standard popup for time inputs. If the user changes either time, the new times will be stored in *startTime* and *endTime* and the function will return a value of 1. If the user does not modify the times, then the values in *startTime* and *endTime* will not change and the function will return a value of 0. The *title* can be used to prompt the user for the type of times being requested. This function requires Symbian OS 3.3 or later. On some devices, calling this function will cause the After Script to be run on return.


Enter appointment time

Start Time: 12:00 am

End Time: 12:00 am

Example
```
Dim apptstart as time
Dim apptend as time
Dim res as integer
Res=PopupTime(apptStart, apptEnd , "Enter the
times of your appointment")
```

Pow Function

POW(*theNumber* as Float , *pow* as Float)

Description
Returns a float with *theNumber* raised to the *pow* power

Example
```
Dim result as Float
Result=Pow(12.2, 2.5)
```

Power10 Function

POWER10(*theNumber* as Float , *pow* as Integer)

Description
Returns a float with *theNumber* raised to the *pow* power of 10

Example
```
Dim result as Float
Result=Power10(12.2, 3) 'would result in an answer of 12,200
```

Proper Function

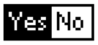
PROPER(*theString* as String)

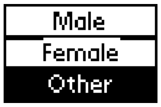
Description
Returns a new string with the 1st character of each word of *theString* capitalized)

Example
```
Dim result as String
Result=Proper("john doe") 'would set result to "John Doe"
```

PushButton Object

A PUSHBUTTON is a box containing text that is either "not selected" and therefore printed in normal black-on-white text mode, or "selected" in which case it is printed in reverse with a black background and white text (Highlighted). The user changes a PushButton from selected to not selected or vice versa by tapping on it. PushButton objects are normally used to indicate a choice of some parameter or value where the possible choices are either yes/no, on/off, etc. Use the Group ID property to make PushButtons mutually exclusive.





Properties Supported (Set at design time)
Label, Font ID, Text, Group ID, Anchor Left, Selected, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height ,Hide, Show, Redraw, Text, Status, ID, Index, Type

Example
```
Msgbox "The PushButton status is " + str(selYesNo.status)
if mydb_yes_no_fld=1 then selYesNo.status = nsbOn else selYesNo.status = nsbOff
```

RadToDegrees Function

RADTODEGREES(*theAngleInDegrees* as float)

Description
Returns a float with an angle in radians converted to degrees.

Example
```
x=RadToDegrees(y)
```

Rand Function

RAND()

Description
Calculates a random float in the range of 0 to 1.0

Example
```
Dim result as Float
Result=Rand() 'will set result to some random number from 0 to 1.0
```

Redraw Statement

REDRAW

Description
Causes the current display form to be redrawn to refresh the contents. This statement cannot be used in the Form Before Code, as the form has not be drawn yet. It causes the form's Form Before and Form After scripts to be run. It should be the last statement in a subroutine, since the form is about to be completely redrawn.

Example
```
Redraw
```

Rem Function

REM(*a* as double, *b* as double)

Description
Returns a float with the remainder of one number divided by another.

Example
```
Dim a as double
Dim b as double
Dim x as double
X=rem(a, b) 'if a was 2.457 and b was 2 then x would be .457
```

Repeater Object

A REPEATER object is similar to a Button except that the repeater will continue to fire the code associated with this object for as long as the object remains selected (as long as the user holds the stylus down on the object). Normally this is used by displaying up or down or left and right arrow symbols from the symbol font as the Button's text. Depressing the up or down arrow can be programmed to increase or decrease the value of a variable and display the new value in a separate field on the form. Used in the built-in applications frequently to continually increase the date's year, or month, or day values.

Properties Supported (Set at design time)
Label, Font ID, Text, Anchor Left, Frame, Non-bold Frame, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Redraw, Text, ID, Index, Type

Return Statement

RETURN

Description
Transfers control to the statement following a GOSUB statement.

Example
```
GoSub Rtn1
....
Rtn1:
....
Return
```

Right Function

RIGHT(*theString* as String, *num* as Integer)

Description
Returns the rightmost *num* characters of *theString*. If *theString* has less than *num* characters, the result will be less than *num* characters.

Example
```
Dim result as String
Result=Right("John Doe", 6) 'would set result to "hn Doe"
```

RightPad Function

RIGHTPAD(*theString* as String, *num* as Integer)

Description
Returns a string that is *num* characters long. If the input string, *theString*, is less than *num* characters long, then spaces are appended to the value of *theString* to pad on the right.

Example
```
Dim result as String
Result=RightPad("John Doe", 12) 'would set result to "John Doe    "
```

Round Function

ROUND(*theNumber* as Float , *pos* as Integer)

Description
Returns a float rounding the number in the *pos* decimal position by rounding up if the fraction following the *pos* digit is .5 or more or by truncating if the fraction is less than .5.

Example
```
Dim result as Float
Result=Round(22.1256, 2) 'result would be 22.13
Result=Round(7.12312, 3) 'result would be 7.123
```

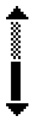
RTrim Function

RTRIM(*theString* as String) As String

Description
Returns a string with all trailing space characters removed from the end of the string.

Example
```
Dim result as String
Result=RTrim(CityName)
```

Scrollbar Object



A SCROLLBAR is a bar that the user can click on to select a position somewhere between the beginning and end of the bar. It is normally used to select a position within some variable number of elements. If the width is greater than the height, the scrollbar will be horizontal.

Properties Supported (Set at design time)
Min Value, Max Value, PageSize, Value, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Min, Max, PageSize, Current, ID, Index, Type

Example
```
Msgbox "Current scrollbar position is " + myScrollbar.Value
```

Second Function

SECOND(*theTime* as Time)

Description
Returns an integer with the seconds value from *theTime.*

Example
```
Dim theTime as Time
Dim result as Integer
theTime=TimeVal(13,24,15)
result=Second(theTime) 'will calculate result=15
```

Select Case / Case / End Select Statement

SELECT CASE *varname*
   CASE *value1*
      Statement Group 1
   CASE *value2*
      Statement Group 2
...
   CASE ELSE
      Statement Group n
END SELECT

Description
To execute specific statements based on the value of variable *varname*. The CASE ELSE is optional. If supplied, it specifies the statements to be executed if varname has any values except those identified in previous CASE value statements. The values in CASE statements must be variables or literals. Expressions are not allowed.

Example
```
Dim PolicyType as String
.....
Select Case PolicyType
  Case "Auto"
    CalcAutoPremium
    ShowAutoPolicy
  Case "Home"
    CalcHomePremium
    ShowHomePolicy
  Case Else
    CalcGenericPremium
    ShowGenericPolicy
End Select
```

Selector Object

A SELECTOR object is an object that displays a text string enclosed in a gray rectangular frame. Normally used to show date or time. This is normally programmed such that when the user taps on this object, the associated code is executed and it will bring up a date or time popup where the user can enter a new date or time. The actual width of the selector object is automatically set at runtime from the text and the font.

Properties Supported (Set at design time)
Label, Font ID, Anchor Left, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Redraw, Text, ID, Index, Type

Example

SerialClose Function

SERIALCLOSE()

Description
Closes the serial port to discontinue its use. The serial port must be open. Returns an integer error code. See SerialOpen for a list of result codes.

Example
```
Err=SerialClose()
```

SerialOpen Function

SERIALOPEN(*port* as integer, *baudRate* as integer)

Description
Opens the serial port to prepare for subsequent use for input and/or output. *Port* is the serial port number, which will normally be zero. See table below for other values *BaudRate* is the transmission speed to communicate at: all standard baud rates to 57600 have been tested. See Tech Note 16 for more information on Communications.

Port Values

0	Cradle Port (USB or RS-232)
32768	Cradle Port (USB or RS-232)
nsbSIR	IR – Raw IR (also called SIR), no protocol support
32771	Cradle - RS-232
32772	Cradle - USB
nsbBTCM	'btcp' Bluetooth Connection Manager
nsbIRCOMM	'ircm' IR Comm Protocol.
nsbRFCOMM	'rcfm' Bluetooth Virtual port

Physical and virtual port number are also allowed.

Serial Result Codes

0	Operation successful
1	Operation failed

2	Internal state machine error
104	Not enough memory to open port
769	Bad Parameter
770	Bad Port
771	No Memory
772	Bad Connect ID
773	Time Out
774	Line Error
775	Already Open
776	Still Open
777	Not Open
778	Not Supported
779	No devices available
780	Configuration failed
1286	Time Out
4353	No dial tone
4354	No carrier/tmeout
4355	Busy signal
4356	Cancelled by user
4357	Command error
4358	No modem detected
4359	Not enough memory
4360	Modem prefs not set up
4361	Dial command error: dial string too long or bad characters
4362	No phone number and not :"direct connect"

Example

```
Err=SerialOpen(0,28800)
```

SerialReceive Function

SERIALRECEIVE(*buffer* as string or integer arrray, *numChars* as integer, *timeout* as double)

Description
Receives incoming data from the serial port and places it into *buffer*. It can be used to wait for data to arrive, or to get data once a program has been notified of incoming data by an NsbSerialIn event. It stops after receiving at least *numChars* characters and cancels further receipt of data if more characters are received than the BufferSize set with the SerialSet() function. This function returns a result code: see SerialOpen for a list of result codes. To clear the receive buffer, do a SerialSend("",0).

If SerialReceive is called as a result of an NsbSerialIn event, it will return when all the characters are read, even if *numChars* has not yet been received. The timeout argument is ignored. To find out how many characters are in the serial receive queue, using the SysInfo(5) function.

If SerialReceive is not called after an NsbSerialEvent, it will return when *numChars* are received or no characters are received in the *timeout* period in seconds. Fractions of a second are allowed. In the event of a timeout, the function will return 773, along with any data that was received in buffer.

If buffer is an integer array, the ASCII value of each byte (including null) will be put into an element of the array. This allows you to receive null characters.

Example
```
Err=SerialReceive(inputString, 200, .5) 'non event case
```

SerialReceiveWithEvent Function

SERIALRECEIVEWITHEVENT()

Description
This function sets up the device to wait for data to come in, When characters arrive, an NsbSerialEvent is generated that you can pick up in your Event code. This function has no parameters. See SerialOpen for a list of result codes. This feature may not work on all devices: the solution is to use a timer to check the serial port regularly and use SerialReceive.

Example
```
'anywhere in your program after serial port is opened
Global err as integer
err=SerialReceiveWithEvent()'in your form's Event Script
   Dim received as String
    If getEventType()=nsbSerialIn Then
        MsgBox "Serial In Event"
        err=serialReceive(received, 10, 0)
      If Err > 0 Then
        Beep
        error.text=str(err)
        data.text=data.text+"."+Received
      Else
        error.text=""
        data.text=data.text+received
      End If
    End If
```

SerialSend Function

SERIALSEND(*buffer* as string or integer array, *noChars* as integer)

Description
Transmits data out over the serial port. *Buffer* is the string of data to be sent. If *buffer* is an integer array, each element (including zero values) will be sent as an ASCII character. *NoChars* is the number of characters from *buffer* to send. Returns an integer error code. See SerialOpen for a list of result codes. To clear the receive buffer, do a SerialSend("",0).

Example
```
Err=SerialSend(outString, Len(outString))
Err=SerialSend(chr(0),1) 'send a null character
```

SerialSet Function

SERIALSET(*characteristic* as string, *value* as integer)

Description
Sets a communication *characteristic* to control serial I/O. Must be done after SerialOpen. Returns an integer error code. See SerialOpen for a list of result codes.

		default
Autobaud	0=off, 1=on	1
Buffersize	Maximum number of characters for serialReceive.	512
Baudrate	Speed of communication in bits.(e.g. 28800)	SerialOpen
BitsPerChar	5,6,7,or 8	8
Cmdtimeout	Timeout in microseconds	500000
Ctsauto	1 for on, 2 for off	1
Ctstimeout	number of seconds before timeout error occurs	5
Dcdwait	Seconds to wait for connection	70
Dialtone	0 for pulse dialing, 1 for tone dialing	1
DtrAssert	0=normal, 1=asserted	0
Dtwait	Seconds to wait for dialtone	4
Handshake	Modem handshaking 0=off, 1=on	0
IR	0=no IR, 1=Receive IR, 2=Send IR	0
Parity	1 for odd parity, 2 for even parity, 3 for no parity..	3
PortID	Set the serial PortID to a new value. Can be used to control more than one serial connection (if hardware allows it) See SysInfo(3)	0
Rtsauto	1 for on, 2 for off	1
Stopbits	the number of stopbits to use(1, or 2)	1
Volume	a level from 0 to 3 where 0 is off and 3 is maximum	1
XonXoff	1 for on, 2 for off	1

Example
```
err = SerialOpen(32769,9600) ) 'Set up for IR communications
err = SerialSet("IR",1)
```

SetCurrentWindow Statement

SETCURRENTWINDOW *winName* as String

Description
Specifies the graphics window which is to become the active graphing window. All subsequent graphing commands apply to this window. The window must have been previously defined by a CREATEWINDOW command.

Example
```
SetCurrentWindow "barGraphWin"
```

SetEventHandled Statement

SETEVENTHANDLED

Description
To cause the Device operating system to ignore the special event. This command should only be issued in special event code subroutines. If you use call SetEventHandled, then the Symbian OS will not process the event. This includes the power on/off key. You should use great caution in your handling of the power on/off key as you could prevent the machine from powering off if you do not let the operating system handle this key.

Example
```
SetEventHandled
```

SetTheme Function

SIGN(*themeName* as String)

Description
Changes the color theme of objects and background. *themeName* must be included as a resource in the project. If successful, it returns 0, an error code otherwise. A number of themes are included with NS Basic/Symbian OS in the Themes directory. Themes can be created and edited using the Theme Editor (see 04.J).

Example
```
Dim result as Integer
Result=SetTheme("Classic") 'colors of objects and backgrounds will change
```

Shift Indicator Object

The SHIFT INDICATOR object is an indicator that can appear based on the input mode. Otherwise, the shift indicator does not appear on the form. If you desire a shift indicator, you must place this object on your display form at design time.

Properties Supported (Set at design time)
Left, Top

Methods Supported (See "Methods")

Sign Function

SIGN(*theNumber* as Float)

Description
Returns 1 if *theNumber* is 0 or positive or –1 if *theNumber* is negative.

Example
```
Dim result as Integer
Result=Sign(-2.5) 'result would be -1
```

Sin Function

SIN(*theAngle* as float)

Description
Returns a float with the sine of the arithmetic expression argument. The angle must be expressed in radians.

Example
```
y=DegToRadians(180)
x=Sin(y) will result in the sine of 180 degrees
```

Sinh Function

SINH(*theAngle* as float)

Description
Returns a float with the hyperbolic sine of the arithmetic expression argument. The angle must be expressed in radians.

Example
```
y=DegToRadians(180)
x=Sinh(y) 'will result in the hyperbolic sine of 180 degrees
```

Slider Object

A SLIDER object represents a value that falls between a particular range. For example, a slider might represent a value between 0 and 10. *Page Size* is the increment that the *Value* changes when the slider is moved.

Properties Supported (Set at design time)
Maximum, Minimum, Page Size, Value, Visible

Methods Supported (See "Methods")
Left, Top, Width, Height, Hide, Show, Redraw, ID, Index

Example
```
Record=mySlider.Value / mySlider.Width * dbGetNoRecs(myDatabase)
```

Sound Statement

SOUND *freq* as integer, *duration* as integer, *amplitude* as integer[, *wait*]

Description
Plays a sound through the Device speaker. *Freq* is the frequency in Hertz. *Duration* is the duration in milliseconds. *Amplitude* is the sound amplitude. Allowed values depend on the device. Values you can use are 0,2,4,8, 32 and 64. *Wait* sets the wait mode. If *Wait* is nsbWait, program execution is halted until the sound is completed. If Wait is nsbNoWait program execution proceeds while the sound is played. nsbNoWait is the default if neither is specified. As of Symbian OS 3.5, *wait* is not implemented.

Example
```
Sound 1200, 1000, 2, nsbWait
```

Sqrt Function

SQRT(*theNumber* as Float)

Description
Returns the square root of a number as a float.

Example
```
Dim result as Float
Result=Sqrt(16.0) 'result would be 4.0
```

Stop Statement

STOP

Description
End execution of the application and returns to the Launcher app.

Example
```
Stop
```

Str Function

STR(*theNumber* as Float)

Description
Returns a string by converting *theNumber* from internal floating point to a character string. Negative numbers will be preceeded by a minus sign(e.g. "-12.5")

Example
```
Dim result as string
Dim x as Float
X=-22.56
Result=Str(x) 'result would be the string "-22.56"
```

Sub Statement

SUB(*varList*)

Description
Define the start of a subroutine and the variables that are passed as arguments. Each variable in *varList* is defined as *Varname* AS varType where *varType* is the type of variable (Integer, Float, Array, UDT etc, except for Database). Variables are separated by commas. A SUB statement defines the start of a subroutine and the end of the subroutine is defined by the END SUB statement. As opposed to functions, a subroutine does not return a value. Results must be returned by setting the value of one or more argument variables. The execution of END SUB statement returns control to the calling program. However, an EXIT SUB statement immediately terminates the function and returns control to the calling program.

Example
```
Call CalcSub(5.0 , 50000, interest)
.......
Sub CalcSub(rate as Float, amount as Float , int as Float)
  Int=rate / 100 * amount
End Sub
```

SubtractDays Function

SUBTRACTDAYS(*theDate* as Date, *Days* as Integer)

Description
Subtracts days from date and returns a new date value.

Example
```
Dim theDate as Date
Dim newDate as Date
TheDate=ToDate("01/01/97")
newDate=SubtractDays(theDate, 15) 'subtracts 15 days

newDate would now contain a date of 12/17/96
```

SubtractMonths Function

SUBTRACTMONTHS(*theDate* as Date, *Months* as Integer)

Description
Subtracts *Months* from *theDate* and returns a new date value.

Example
```
Dim theDate as Date
Dim newDate as Date
TheDate=ToDate("06/01/98")
newDate=SubtractMonths(theDate, 15) 'subtracts 15 months
newDate would now contain a date of March 1, 1997
```

SubtractYears Function

SUBTRACTYEARS(*theDate* as Date, *Years* as Integer)

Returns: A new date

Description
Subtracts *Years* from *theDate* and produces a new date value.

Example
```
Dim theDate as Date
Dim newDate as Date
TheDate=ToDate("03/01/98")
newDate=SubtractYears(theDate, 4) 'subtracts 4 years

newDate would now represent a date of March 1 , 1994
```

SysEventAvailable Function

SYSEVENTAVAILABLE()

Description
See if a low level system event (such as a pen or key event) is available. A use of this function is to use it as a way to let the user signal to interrupt a continuous screen display. For example, the sample program "digiclock" continuously redisplays the screen with the latest time-of-day. In the screen display routine, the SysEventAvailable routine is used to test if the user has tapped the screen and if so, the display routine is terminated. Returns 1 if there is an system event pending and 0 if there are no system events pending. See GetEventType and GetKey for more information on event handling.

Example
```
If SysEventAvailable()=1 then goto quitDisplay
```

SysInfo Function

SYSINFO(arg)

Description
Return system information.

0	Runtime version
1	Number of clock ticks
2	Ticks per second for current device
3	The PortID of the current serial connection. Use with SerialSet("PortID",x)
4	The current FormID
5	Number of bytes currently in the serial receive queue
6	Symbian OS ROM Version
7	CoordinateSystem. 72 for LoRes, 144 for HiRes
8	KBytes of unused storage heap. Check this before doing memory hungry operations.
9	This is the character code that caused the event. This could be any of the regular keyboard characters, a soft or hard button on the device or other event.
10	This value is used to contain additional information about the chr. For example, the 5-Way button uses this value to tell if the event is up, down, left or right.
11	You will need to test the bits of this value to check if a flag is set. For example, 0x0008 being set means the 'commandKeyMask' bit is set and the chr received is a command, not a character.

Example

```
If SysInfo(0)<400 then MsgBox "Please install latest version of the Runtime."
```

SysInfoSet Function

SYSINFOSET(*characteristic* as string, *value* as integer)

Description
Sets a system *characteristic* to *value*.

		default
CoordinateSystem	Set to 72 for LoRes, 144 for HiRes. See Section 10.J.	
HeapVariableCount	Sets the number of numeric variables to be allocated in the Dynamic Store, where n is the count of variables (default is 200.) This results in faster execution, since variables can be more quickly accessed. Variables are added to this area in order of compilation. Each variable used takes 22 bytes of space from the Dynamic Store. Consult manufactuer's documentation to find out how much Dynamic Store is on your device. Only the actual number of variables used is allocated. The full number in HeapVariableCount will only be used if you actually have that number of variables. This function should be called in your Program Startup code.	200

Timer	Timer feature allows you to generate your own events that fire after a set interval. This is useful for situations where you expect a response after a certain interval and you want to take action if you do not get that response. For example, if you are doing communications with another computer and you want to check from time to time if any data has come in.
	To set a timer, do the following statement:
	`SysInfoSet("Timer", 3000)   'Set a timer in 3 seconds.`
	To respond to the Timer event in your program, use something like this in your Events script:
	`If getEventType()=nsbTimer then`
	`  MsgBox "Timer Event Received" 'respond to event`
	`  SysInfoSet("Timer", 3000)     'set another timer`
	`End If`
	To cancel a timer currently in effect, do `SysInfoSet("Timer", 0)`. If you exit your program, the timer is cancelled. There is no need to do SETEVENTHANDLED for this event. See the Timer sample to try this out.	

Example

```
SysInfoSet("HeapVariableCount",200)
```

SysTrapFunc Function

SYSTRAPFUNC(*trapnum*, *numargs*[, *arg1*[, *arg2*[, ...]]]))

Description
Trapnum is a constant, defined in Symbian OS, that determines which system routine is being called. *Numargs* is the number of arguments that are passed to the system procedure. Additional arguments that are included will vary, depending on which system procedure is called. Special attention must be paid to the types of arguments and return values that a system procedure uses. Please read Tech Note 6 for the complete information on this function.

Example

```
Res=SysTrapFunc(531, 4, 80, 0, 80, 160)  'Calls trap 0xA213 (WinDrawLine)
```

SysTrapSub Function

SYSTRAPSUB(*trapnum*, *numargs*[, *arg1*[, *arg2*[, ...]]]))

Description
Trapnum is a constant, defined in Symbian OS, that determines which system routine is being called. *Numargs* is the number of arguments that are passed to the system procedure. Additional arguments that are included will vary, depending on which system procedure is called. Special attention must be paid to the types of arguments and return values that a system procedure uses. Please read Tech Note 6 for the complete information on this function.

Example

```
SysTrapSub 531,4,80,0,80,160   'Calls trap 0xA213 (WinDrawLine)
```

Tan Function

TAN(*theAngle* as float)

Description
Returns a float with the tangent of the arithmetic expression argument. The angle must be expressed in radians.

Example

```
y=DegToRadians(180)
x=Tan(y) will result in the tangent of 180 degrees
```

Tanh Function

TANH(*theAngle* as float)

Description
Returns a float with the hyperbolic tangent of the arithmetic expression argument. The angle must be expressed in radians.

Example

```
y=DegToRadians(180)
x=Tanh(y) will result in the hyperbolic tangent of 180 degrees
```

TestNum Function

TESTNUM(*theString* as string, *SignOption* as string, *NoDigitsBeforeDecPt* as integer, *NoDigitsAfterDecPt* as integer)

Description
Tests a string variable's contents to see if :
1. The string represents a valid numeric number.
2. If SignOption is " "(blank) then the string must not have a leading + or – sign. If SignOption is "+" or "-" then a leading + or – sign is allowed.
3. The number of numeric digits in the string appearing before the decimal point cannot exceed *NoDigitsBeforeDecPt.*
4. The number of numeric digits in the string appearing after the decimal point cannot exceed *NoDigitsAfterDecPt*.

Returns 0 if all tests are met successfully or a non-zero if tests fail.

Example
```
Dim result as integer
Dim temp as string

Temp=fldVoltageLevel.text 'picks the string from the field object

'the following test allows a + or – string and 3 digits before decimal point
' and 2 digits after the decimal point
If Not TestNum(temp, "-" , 3, 2)=0 then
  Msgbox "The voltage has been entered incorrectly"
End if
```

TimeDiff Function

TIMEDIFF(*timeVar1* as Time, *timVar2* as Time)

Description
Returns the number of seconds between *timeVar1* and *TimeVar2*.

Example
```
Dim timeVar1 as Time
Dim timeVar2 as Time
Dim result as integer
TimeVar1=ToTime("06:08:01")
TimeVar2=Now()
Result=TimeDiff(timeVar1, timeVar2)
```

TimeVal Function

TIMEVAL(*theHour* as Integer, *theMinute* as Integer, *theSecond* as Integer)

Description
Returns a Time type result by converting the hour, minute, and second inputs to Time format. Hours should be in the range of 0 to 23.

Example
```
Dim result as Time
Result=TimeVal(14,38,45) 'will set result to the time of 14:38:45(2:38:45 PM)
```

ToDate Function

TODATE(*theString* as String)

Description
Converts *theString* from the format "YYYY/MM/DD" or "YY/MM/DD" to an internal date result and returns it.

Example
```
Dim result as Date
Result=ToDate("1998/06/13") 'would set result to 1998/06/13
```

Today Function

TODAY()

Description
Returns the the current date extracted from the computer's internal date and time clock.

Example
```
Dim result as Date
Result=Today() 'would set result to today's date
```

ToTime Function

TOTIME(*theTime* as String)

Description
Return the string *theTime* in the format "HH:MM:SS" converted to an internal time result.

Example
```
Dim result as Time
Result=ToTime("14:38:25") 'would set the result to 2:38:25 PM
```

Trim Function

TRIM(theString as String) As String

Description
Removes all leading and trailing space characters from the start and end of the string and returns it.

Example
```
Dim result as String
Result=Trim(CityName)
```

Trunc Function

TRUNC(theNumber as Float , noDgts as Integer))

Description
Truncates the input number to a desired number of decimal places without rounding of the result and returns a float.

Example
```
Dim result as float
Result=Trunc(123.5678, 3) 'would set result to 123.567
```

Type Statement

TYPE userName

Description
To begin the definition of a user-defined definition of a data type or structure.

Example
```
Type PersonType
  Name as string
  Occupation as string
  Salary as double
End Type

Dim person as PersonType
Person.name="John Doe"
Person.Occupation="Freelance artist"
Person.Salary=50000.00
msgbox "The person's name is " + person.name
msgbox "The person's occupation is " + person.occupation
msgbox "The person's slary is " + str(person.salary)

There may also be arrays of the user-defined type.
Dim workers(100) as PersonType
Sum=0
For I=1 to noWorkers
  Sum=Sum + workers.salary(i)
Next
Avg=sum / noWorkers
```

Ucase Function

UCASE(*theString*)

Description
Returns a string with the upper case conversion of the input string

Example
```
Dim result as String
Result=UCase("john doe") 'would set result to "JOHN DOE"
```

Val Function

VAL(*theString* as String)

Description
Returns a text string to be converted to a float.

Example
```
Dim result as float
Result=Val("-12.456") 'result would have the value -12.456
```

Year Function

YEAR(*theDate* as Date)   Function

Description
Returns an integer which is the year value from the Date variable *theDate*

Example
```
Dim theDate as Date
Dim result as Integer
theDate=DateVal(1995,6,23)
result=Year(theDate) 'will calculate result=1995
```

YearMonth Function

YEARMONTH(*theDate*)

Description
Returns an internal Date type converted into a text string

Example
```
Dim theDate as Date
Dim result as String
TheDate=DateVal(1995,6,23)
Result=YearMonth(theDate) 'will give the string "95/06"
```

08. File Handling

NS Basic/Symbian OS uses two methods to access data in files. The first, described in this section, uses pdb files to store data. This is a fairly easy to use file system that allows both sequential and random access to data. An index allows quick lookup of data records. The file format is the same as for Palm OS Databases, which is why files are sometimes referred to as "databases".

The second method of access files is as flat files. Tech Note 23 "Using the File Library" discusses how to use this method. This library lets you create and access files in the public storage area. It also lets you copy files from the public area to your own program's private area and back. It also lets you access files in multiple formats, including txt, csv and indeed, any data structure.

A number of third parties have created desktop programs to convert data between PDB, Access, CSV, Excel, and other formats Be sure to check out the NS Basic/Symbian OS support group and ask around. You'll find lots of useful information on this topic there.

The Symbian OS doesn't much care what you put in a record. A record is just a series of bytes. Your app has to interpret those bytes in whatever way you want it to.

Samples for this chapter are included in the Projects folder that gets installed with NS Basic/Symbian OS. For the latest information on pdb files, check Tech Note 2 on the NS Basic website.

08.A Reading and Writing to non-Keyed Files

This method of file access is probably the most simple in concept, but can often cause the most confusion. Think of a file in NS Basic/Symbian OS as a sort of 'grid'. Each 'cell' in the grid holds one byte of data. The description of the DIM statement in the NS Basic/Symbian OS Handbook has a list of the NS Basic/Symbian OS data types and their storage requirements. Strings are an exception. They require one byte perform character plus an additional byte for their terminator. So writing the word 'Cat' to the beginning of a record would look something like this (the '\0' symbol denotes the end of the string):

C	a	t	\0								

Writing to a non-keyed file is simple. You just need to be mindful of where you place your data. Dates and Numbers, for example, are easy enough, as they'll translate to 8-byte values (for dates and times) and 4-byte values (for Integers and single-precision floating point values). When you put Strings in the mix you need to force them to the length that you want your data to be in, otherwise you'll fire off DmWriteCheck exceptions. If your field that contains the word Cat could be up to 15 characters in length what you store would look something like this in the memory:

C	a	t													\0				

To write to a non-keyed file you need to use two different functions: dbPosition() and dbPut(). The dbPosition function will set your record pointer to the specific record number and 'offset' (or 'cell' as discussed above). When data is written using dbPut() the pointer into the record will be set to the end of the last piece of data written. In this example a name exists in a field object called fldName. This code will write the contents of fldName to the beginning of the first record of the file.

```
'assume name is DIMed as String and db as file
name = rightpad(fldName.text,15) 'set the string to be 15 characters long.

res = dbposition(db, 1, 0)
res = dbput(db,name)
if res not = 0 then
     msgbox "Write error: " + str (res)
end if
```

To write more data to the same record is simple. Just use more dbPut statements.

```
'assume that dDate is DIMed as a Date type
res = dbput(db,dDate)
if res not = 0 then
     msgbox "Write error: "+str(res)
end if

'assume fmyFloat is DIMed as a Float
res = dbput(db,fmyFloat)
if res not = 0 then
     msgbox "Write error: "+str(res)
end if
```

To read these back is just about as simple. Just use dbPosition() along with dbGet(). Each call to dbGet will move the pointer in the record, so there's no need to use repeated calls to dbPosition (although you can if you only want particular fields from a record in the file).

```
res = dbposition(db, 1, 0)
res = dbget(db,name)
if res not = 0 then
     msgbox "Read error: "+str(res)
end if
fldName.text = rtrim(name) ' removes the spaces padded on by the rightpad()
function.

dDate=-1
res = dbget(db,dDate)
if res not = 0 then
     msgbox "Read error: "+str(res)
end if
fldDate.text = datemmddyy(dDate) 'convert the date into a displayable format

res = dbget(db,fmyFloat)
if res not = 0 then
     msgbox "Read error: "+str(res)
end if
fldCost.text = str(fmyFloat)
```

Most desktop utilities that produce .pdb files create them without sorted keys. To work properly as a sorted file, the first field of each record must be in sorted order. Be sure to use the "Operations by Position" functions to access them, not "Position by Key."

08.B Reading and Writing to Keyed Files

Writing to a keyed file is a matter of having a unique ID for each record (called a 'key'). The NSBasic file header maintains the key information along with the offsets into the file where the information is stored. The two biggest advantages to this are ease of access and ease of storage. To find a record in a keyed file you need only search by the key value (in a non-keyed file you need to search one record at a time). With keyed files you can also store your information using a User Defined Type or UDT (created with the NS Basic/Symbian OS TYPE statement). This way instead of writing say, 50 elements, into a file one element at time (as with dbPut) you have a single dbInsert() statement with 3 arguments:

```
dbInsert(database,keyvalue,UDT)
```

Here's an example. Like the non-keyed file example above, this code will write one string, one date and one floating point number into a PDB file. Assume the following UDT:

```
type myData
     name as string
     dDate as Date
     fmyFloat as float
End type
```

Next, assume the following variables are defined in your Project Startup code:

```
Global db as Database
Global msgdata as string
Global recordData as myData
```

Writing to the file with the dbInsert command is as simple as this (this assumes the file has been created and is currently open):

```
recordData.name = "John Doe"
recordData.dDate = mmddyytodate("09/29/99")
recordData.fmyFloat = val("123.45")

Dim res as Integer
Dim keyval as Integer

Keyval = 1
res=dbinsert(db,keyval,recordData)
```

This writes a new record to the file with a key of 1 and now holds the data shown above. Reading the data back out is almost exactly the same, just in reverse:

```
Keyval = 1 ' must be set to the key of the record you want to retrieve

res = dbread(db,keyval,recordData)
if res = 0 then
      msgbox recorddata.name
      msgbox datemmddyy(recorddata.dDate)
      msgbox str(recorddata.fmyfloat)
else
      alert("Oops!","An error occured : " + str(res),3,"OK")
end if
```

If you get a return value of 2 in 'res', this means that the record you tried to read wasn't found and the next closest record to it was returned instead.

09. Running and Distributing your App

When you have finished coding your app, you need to compile it from the Run menu. At this point, the syntax of your program is checked If a problem is found, the compile step stops, an error message appears, and the Code Window may open and highlight the line with the error. Correct it, and start the compile again.

If the compile is successful, installers will be created and run, depending on the options set in Tools…Options. To install a program, your system has to be connected to a Device with the Nokia PC Suite (or equivalent installed). Once installed, you may run the app from the Symbian menu: it will be in the "Installat." folder unless you specified a different folder in your Project Properties.

Apps created using NS Basic/Symbian OS can be distributed free of any royalty to be paid to NS BASIC Corporation or StyleTap Inc. You are of course welcome to sell the apps you create as you see fit.

09.A SIS and SISX Installer Files

Devices have software installed on them using SIS and SISX format install files. NS Basic/Symbian OS creates these files automatically after the compile is complete. Internally, NS Basic/Symbian OS converts the app's icon from .SVGT to .MIF format and creates an .sis and .sisx file. SIS files are usually for UIQ and SISX for S60 devices.

The default signing used is Symbian Self Signed. Symbian Self Signed apps will run on all devices where Software Installation is set to "All" instead of "Signed Only" in the App Manager. For more information on application signing, see Tech Note 11. You will need NS Basic.Symbian OS Pro Edition to sign at a higher level.

If your app uses Libraries or additional files, you will want them to be installed at the same time. Add them to your project under "Resources" in the File Explorer. The compiler will then add them into the installer.

09.B Compiling from the Command Line

You can compile programs from the Windows command line. This allows you to put a compile of your app into a batch script. Here is the format of the command line:

<NS Basic> <program name> -compile

Example

```
"c:\program files\nsbasic\symbian\NSBasic.exe"
c:\nsbasic\projects\samples\derby.prj -compile
```

10. Programming Tips

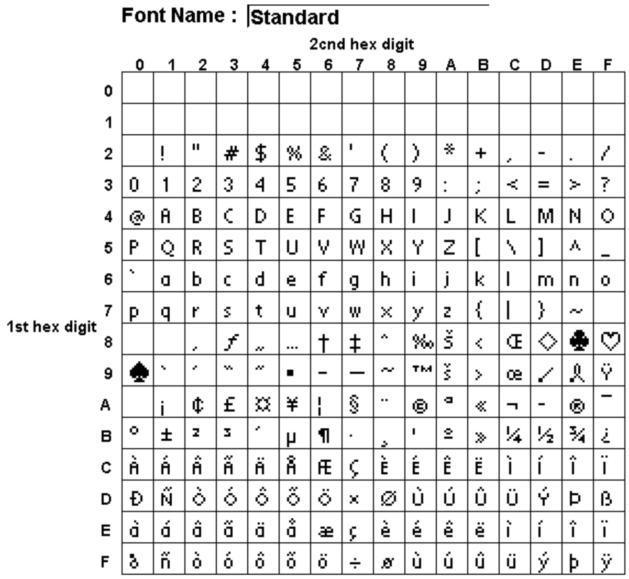
10.A Fonts

Eight fonts are standard with NS Basic/Symbian OS. A font is identified by its font number from 0 to 7. The standard fonts are:

0=Standard
1=Standard Bold
2=Large
3=Symbol
4=Symbol 11
5=Symbol 7
6=LED
7=Large Bold

The font characters are shown in the font diagrams below:

10.A.1 Font # 0



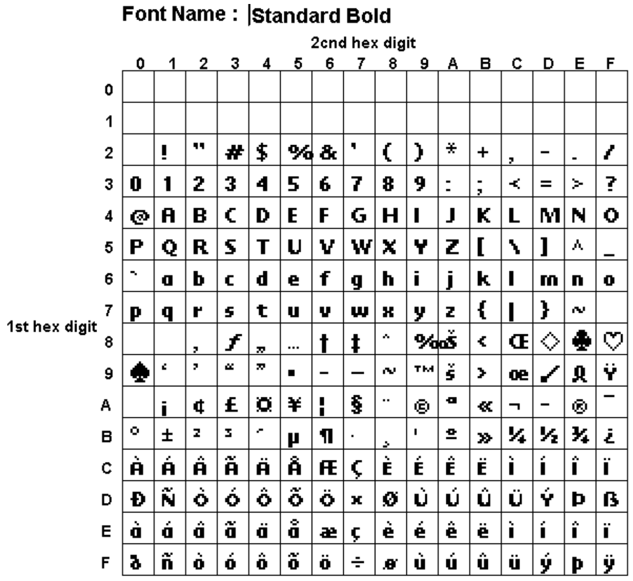
Font Name : Standard

2cnd hex digit

1st hex digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8			‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	◇	♣	♡
9	♠	‘	’	“	”	▪	–	—	˜	™	š	›	œ	✓	ᴥ	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

10.A.2 Font # 1



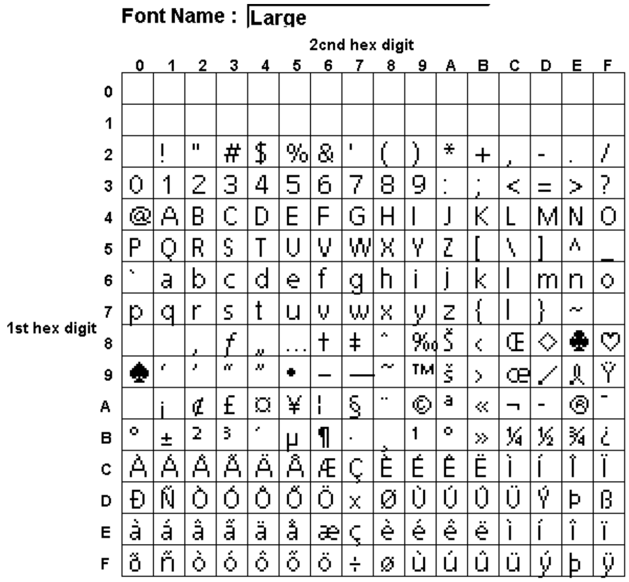
Font Name : Standard Bold

2cnd hex digit

1st hex digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8			‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	◇	♣	♡
9	♠	‘	’	“	”	•	–	—	˜	™	š	›	œ	✓	Ջ	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

10.A.3 Font # 2



Font Name : Large

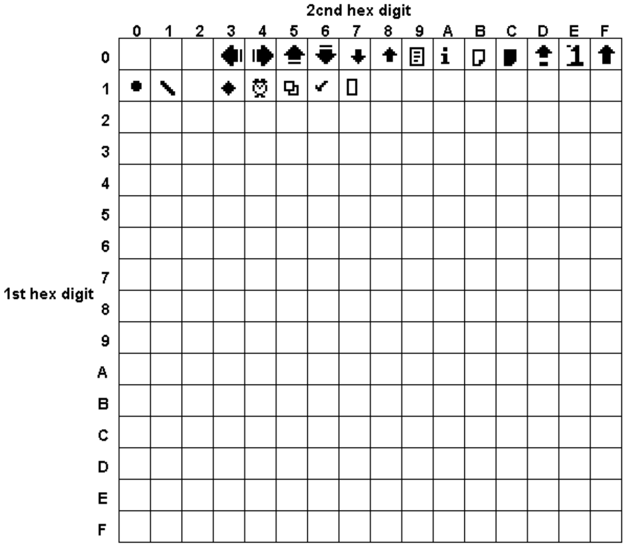
2cnd hex digit

1st hex digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8			‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	◇	♣	♡
9	♠	‘	’	“	”	•	–	—	˜	™	š	›	œ	✓	Ջ	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

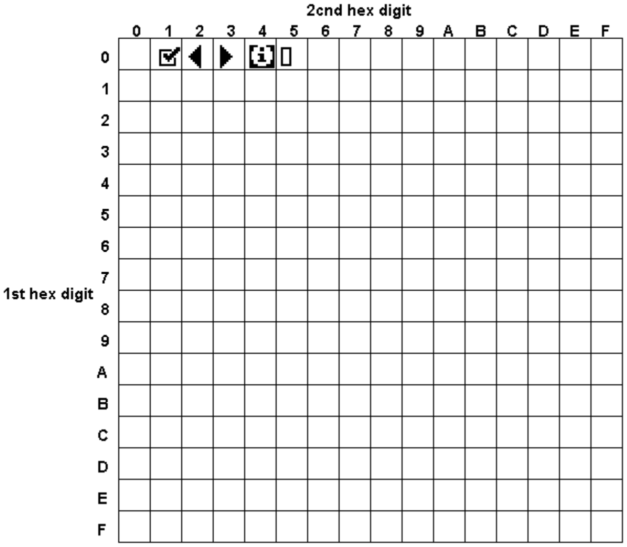
10.A.4 Font # 3

Font Name : Symbol


2cnd hex digit
0 1 2 3 4 5 6 7 8 9 A B C D E F
1st hex digit
0 1 2 3 4 5 6 7 8 9 A B C D E F

10.A.5 Font # 4

Font Name : Symbol 11


2cnd hex digit
0 1 2 3 4 5 6 7 8 9 A B C D E F
1st hex digit
0 1 2 3 4 5 6 7 8 9 A B C D E F

10.A.6 Font # 5

Font Name : Symbol 7

2cnd hex digit

1st hex digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		▲	▼	△	▽	□										
1																
2																
3																
4																
5																
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																

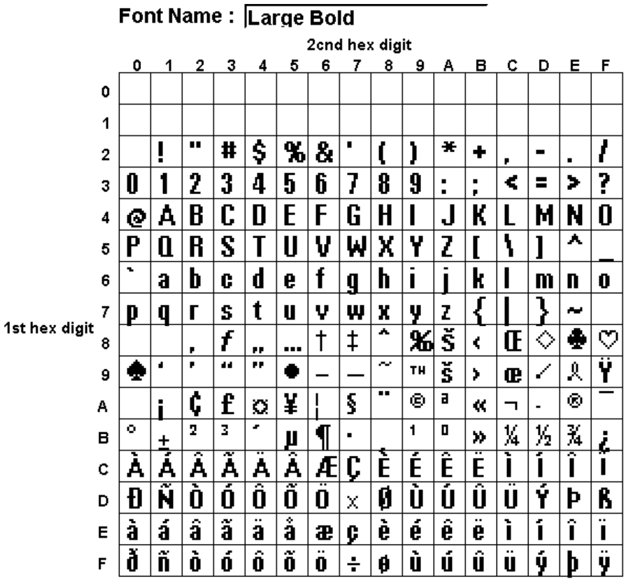
10.A.7 Font # 6

Font Name : LED

2cnd hex digit

1st hex digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2													,	-	.	e
3	0	1	2	3	4	5	6	7	8	9	▯					
4																
5																
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																

10.A.8 Font # 7



Font Name : Large Bold

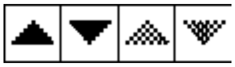
2cnd hex digit

1st hex digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8			‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	◇	♣	♡
9	♠	‘	’	“	”	●	–	—	˜	™	š	›	œ	⁄	⅄	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·		¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

10.B Up/Down Arrows

You can show vertical up or down arrows to give users a choice of what to do(e.g. tap the up arrow to increase a field's value). These frequently are drawn as shown:



If one direction is not appropriate at the current time, then that arrow is drawn grayed-out instead of solid black. NS Basic/Symbian OS has characters in its fonts to draw these arrows both grayed-out and solid black (as shown above). To use these characters, you need to put two label objects on the form where you want the two arrows to appear (usually one above the other for vertical up/down arrows). Each label object should have a name assigned to it such as labUp or labDown. At design time, each label object's text should be set to a 1-character value – it doesn't matter what the value is since it is changed at execution time before the form is drawn. The 1-character label should be a character that shows on the design form so that you know where the object is located on the form ('U' for up arrow and 'D' for down arrow might be helpful). Each object's font should be set to the font from the table below. Then, you need to provide code in the Form After code to assign the hexadecimal codes to change the label object's text to the arrow symbols. The correct hexadecimal codes are:

For Font 5 hexadecimal

&h01	solid black up arrow
&h02	solid black down arrow
&h03	grayed-out up arrow
&h04	grayed-out down arrow

Sample form startup code is shown below:

```
dim up as string
dim down as string
up = &h01
down = &h02
labup.text = up
labdown.text = down
```

This technique can also be used to draw other special characters from any of the NS Basic/Symbian OS fonts.

10.C Programming a Continuously Updating Display

If you need to develop an application that continuously displays some info on the Design Screen without any action or input from the user, (for example, to display a clock with the time-of-day) you can accomplish it by putting your form display code in the Form After code section. Use the Form After code that is triggered after the Device draws any objects. Your code should do any calculations, format any objects, and then use the REDRAW command to force the form to be redrawn. If you wish some time delay between redraws of the form, then you can use the DELAY command before the REDRAW command.

You need to provide code to interrupt the display process, otherwise you will create a never-ending program that requires a reset of the Device. Use code like this in your form display logic:

```
If SysEventAvailable()=1 Then Exit Sub
```

This code exits the display subroutine if the user taps the screen or any button. Usually, the user will tap the application menu button and your program will stop and the standard Symbian OS application launcher will appear.

For examples of this technique, see the DigClock and Derby samples.

Another way to do this is the set a Timer – see SysInfoSet.

10.D Programming Menus

Menus are displayed across the top line of the screen. A program may have several menus, each identified by a menu name. Only one is visible at a given time. An individual menu has items across the top line of the screen called menu bars. Each menu bar may have one or more dropdown menu choices. A dropdown menu choice may have program code associated with it that is activated when the user selects the dropdown choice.

There are no menus displayed automatically by an NS Basic/Symbian OS program. If you want a menu to appear or disappear, you must control these actions within your program. NS Basic/Symbian OS provides the following commands:

MenuDraw	Draws a specified menu at the top of the display screen
MenuErase	Erases the current menu from the display screen
MenuReset	Cancels any dropdown sub-menu that has previously been selected

A good place to draw a menu is in the Form Event code for a form. If you want the same menu to appear for several forms, you should use the MENUDRAW command in the Event code of each form. Here's a sample:

```
If not GetEventType() = NSBKeyorButton then exit sub
If asc(getkey()) = 5 then
        Menudraw "main"
        SetEventHandled
End if
```

MENUERASE will cause the currently displayed menu to disappear entirely.

MENURESET is useful to eliminate the display of dropdown menu choices that may be displayed because of a previous action by the user. MENURESET eliminates any dropdown choices that are displayed, leaving only the top line menu bars on the display screen.

To have a menu separator, use a single dash ("-") as a dropdown item.

There are a number of things that can cause the menu to be unloaded. Here is some sample code using SysTrap calls to make sure the menu stays loaded. Most of the work is done in the Events code section:

```
Dim eventType as Short
Dim menuID as Short
Dim menuP as Variant

eventType = GetEventType()
menuP = SysTrapFunc(450, 0) 'MenuGetActiveMenu
If menuP = 0 Then 'No active Menu
  menuID = 1032 'Menu's ID from IDE
  menuP = SysTrapFunc(445, 1, menuID) 'MenuInit
  SysTrapSub 451, 1, menuP  'MenuSetActiveMenu
  If eventType = nsbKeyOrButton Then
    If asc(GetKey()) = 5 Then 'Menu button
       SysTrapSub 448, 1, menuP 'MenuDrawMenu
    End If
  End If
End If
```

This will trap most of the things that cause a menu to get unloaded; however, a couple of things must also be done to make sure menu shortcuts always work:

1. Put the following in your After code section:

```
SystrapSub 283, 1, "u0=30000:AddEventToQueue"
```

This queues an event to make sure the Menu gets initialized and loaded in the Events code section.

2. For any menu item that doesn't bring up a new form, add this statement to your menu item's click code:

```
SystrapSub 283, 1, "u0=30000:AddEventToQueue"
```

This queues an event to make sure that the Menu gets initialized and loaded in the Events code section.

10.E Programming Mutually Exclusive Objects

There are some objects that can be designed to be mutually exclusive of each other such that if one object is selected, then others must be de-selected. For example, you could have a form with two PushButton objects next to each other for "Male" and "Female". Obviously, only one of these PushButtons should be selected (highlighted) at a time. If the user taps on one on a data entry form, that one should become highlighted and the other one should be unhighlighted. In addition to PushButtons, the same situations can occur for checkboxes. For example, assume you designed a form that had checkboxes for a person's dwelling – separate checkboxes for "apartment" , "rented house", "condominium", or "privately owned residence". Obviously, only one of these should be checked at a time, so checking one should turn off the others.

Both of these situations can be handled by a property of checkbox and PushButton objects called the "Group ID" property. If this property is 0, then an object does not participate in a mutually exclusive group of objects. This is the default setting in the IDE for any new objects you create. However, if a non-zero number is assigned to this property, then this object and any others with the same Group ID value participate in a mutually exclusive group. Checking one object in the group automatically de-selects any other object in the same group. This is done for you, and does not require any actions on your part.

10.F Programming Event Code Routines

Special events are defined to be the press of the menu key, the Up or Down key, or any key generated through the input area strokes in the input area, or a key generated through the use of the onscreen keyboard; or a PenUp or PenDown event caused by pressing the stylus to the screen surface or lifting it from the surface.

Any form may have a special event code subroutine and conversely, only one form is supported by a special event routine, so you must provide separate special event subroutines for every form where you have the requirement to handle special events.

Special event code is optional. The operating system will normally deal with special events but you may choose to handle some or all special events in your own code.

The following are useful commands and functions in special event code subroutines:

GETEVENTTYPE - this function tells you what type of event triggered the special event subroutine
GETKEY - this function will return the value of the last keyed button or character
SETEVENTHANDLED - this command will tell the device operating system that it is to ignore the event. If you do not do this, the operating system will process the event after your subroutine exits. This may be OK and by design, but if you don't want the operating system to process the event, then you must use the SETEVENTHANDLED command.
GETPEN -this command returns the pen coordinates and up/down status.
SYSINFO(9), SYSINFO (10), SYSINFO (11): Check these functions to get the values returned by the operating system after a KeyDown event. These functions provide more information than GETKEY().

Why would you need to provide special event processing?
It's the only way to recognize that the user has pressed special buttons (Menu, Up, Down)
It's a way to get control and do some processing without the need or availability of an object that the user taps on.
You have lower level control of handling keyboard and pen actions.

A good example of the need for this type of code would be if you were writing a program like Paint or Doodle where you wanted to respond to pen or keyboard actions and then modify the screen by drawing with the graphics commands.

If you write programs that continuously display to the screen without user intervention, you need to use the SYSEVENTAVAILABLE function in your form display code to interrupt your display loop if any special user event occurred so that other event codes can be triggered. If you do not, your display loop has continuous control and no other events (even special events) can interrupt your display loop.

10.G Ending A Program

NS Basic/Symbian programs can be ended in two ways: your program can end itself by executing a the `STOP` statement, or it can be terminated by another process or the operating system.

Your program should be designed so it handles being shut down gracefully. Whether your program ends by the user terminating it or you end the program via the `STOP` command, the Project Termination Code is executed. You may use this opportunity to close files or perform other actions as your program is about to terminate its execution.

Next time you start, your Project Startup Code can read the data back in and use that information to go the same state in your program as when it exited.

10.H Signature Capture

NS Basic/Symbian OS can capture signatures and drawings and save them as bitmaps on devices which support touch screens. To do this, first set up a Gadget object on your form, large enough for the signature you wish to capture. If you name the gadget 'Sig', you can then do the following calls:

Sig.displaySignature *string*	Displays *string* in the gadget. Make sure the size of the gadget you use to display a signature is exactly the same size as the gadget you used to capture the signature. Should not be null.
Sig.endSignatureCapure	Turns off signature capture in the gadget's area and returns the current bitmap as a string.
Sig.eraseSignature	Erase any signature that is displayed in the gadget's area.
Sig.startSignatureCapture	Turns on signature capture in the gadget's area.

The format of the returned string is a compressed bitmap, modified so there are no embedded null characters.

The compressed format is formatted as follows:
1-byte count to tell how many times to insert the next byte
1-byte character to insert
1-byte special character (optional--only occurs if the insertion character is hex 01).

If the insertion character is &h01, the special character is hex 01 to insert hex 01. Otherwise, if the special character is &h02, then insert &h00. This sequence repeats until a count of 0 is encountered.

After unpacking the above, the result is a bitmap structure which will vary depending on how many levels of color (pixel depth) were in use on the screen where the capture of the signature occurred. The exact format is not the same as the Windows .bmp format.

Since the signature is stored in a standard String variable, it can be saved and retrieved from a file. For more information, see the sample that is included with the installation.

10.I API Calls

NS Basic/Symbian OS can call System API functions. Operating System APIs, or SysTraps, can be included in an NS Basic/Symbian OS project using SysTrapSub and SysTrapFunc, two NS Basic/Symbian OS keywords. SysTrapSub is used to call system procedures that do not return a value to NS Basic, and SysTrapFunc is used to call system procedures that return a single value to NS Basic.

Documention on API calls is available on the NS Basic website:
http://nsbasic.com/symbian/info/technotes/TN06.htm

10.J HiRes Support

Some devices have HiRes support. These devices allow you to view the screen as 160x160 or 320x320 (HiRes). HiRes applies to Bitmaps and Draw commands.

NS Basic/Symbian OS's HiRes support does not change to how objects are drawn. Objects live in a 160x160 world. When you run on a HiRes device, your objects will be drawn in the same position as before. It is not possible to have your objects automatically drawn at half the size. The advantage of this way of doing things is that your program will look the same, regardless of the screen size.

The Coordinate System setting determines how Draw statements will work. Use SysInfo(7) to get the current Coordinate System setting. In LoRes mode, 72 is returned. In HiRes, 144 is returned. 72 and 144 are arbitrary values, but can be thought of as dots per inch.

Use SysInfoSet("CoordinateSystem",144) to set your device into HiRes mode. The effect on various Draw commands is as follows:

DrawLine: Line will be drawn using 320x320 coordinates. The thickness of the actual line being drawn will be cut in half.

DrawText: The position of the text will be on the 320x320 screen. The characters themselves are unchanged. A different font will be needed to draw smaller text.

DrawRectangle: The position of the rectangle will be based on the 320x320 screen. The line thickness will unchanged.

DrawBitmap: The position of the bitmap will be based on the 320x320 screen. The bitmap itself will be unchanged.

See the sample HiRes.prj to examine how HiRes works.

Signature Capture

This works properly on touch screen HiRes devices. Due to the limit of 64 k in a record, a string or a resource, you will need to limit the size of the input area. It's easy to capture an image larger than 64k. For example, a 100x100 screen capture area on a 16 bit screen is really 200x200 due to HiRes. (200 * 200 * 16 bits)/8 bits per byte = 80,000 bytes, which is too large.

Bitmaps

Bitmaps can have a bit depth of 8 or 16, in both regular and HiRes. The same images that you currently use can be entered as HiRes images, but they will be displayed at half the size of the LoRes version. In other words, if you want to have a HiRes bitmap the same size as a LoRes Bitmap, you need to make it twice as large.

The different images listed under one Bitmap entry in the Project Explorer are collectively called a family. Since the family is saved in a single 64k resource, care has to be taken not to exceed the maximum size. As with Signatures, it is easy to break that limit with just a single image, let alone a family.

If you have a family with both HiRes and LoRes images, the runtime will pick the appropriate one for your device. If the device supports HiRes, the HiRes image will always be chosen, regardless of the Coordinate System being used. If there is no HiRes image and you are on device that supports HiRes, the LoRes image will be displayed at the proper size. For best results, supply both sizes of bitmaps on all projects.

11. Libraries

NS Basic/Symbian OS is able to access standard Libraries. Here are the basics of loading and calling procedures in libraries. See Tech Note 5 for more detailed information.

Procedures in Libraries that return values are called external functions, and procedures that do not return values are called external subs. You should pay attention to the variable types that the procedures call for and try to use the same types in your program. If a string is being returned that is greater than 300 bytes, fill the result variable with enough blanks to handle the returned value before calling.

Libraries are made accessible to the IDE through the use of an Information File (.inf). These files live in the Lib subdirectory of the NS Basic/Symbian OS install directory (ie. C:\NSBasic\Lib\*.inf). To use a Library, your program must first load the library with the LOADLIBRARY statement.

Libraries are installed on the device along with the rest of your app in the SIS installation file. Add the libraries you are using to the Resources section of the File Explorer.

You may write your own Libraries for using tools such as CodeWarrior and the GNU Compiler. For more information, see the Tech Notes at
http://nsbasic.com/symbian/info/technotes/index.htm

NS Basic Corporation makes a number of Libraries available. Here is a list of some of the Libraries that are available. More are being added regularly. See the Tech Notes for the complete list and documention.

BitsNBytesLib	Does bit manipulation, bitwise logical operations, hex and binary conversion and DES encryption. See Tech Note 10 for more information.
ScreenLib	Adds functions to enhance control of the screen. It allows you to change the bit depth and color attributes. There are also drawing functions and a screen lock function. See Tech Note 13 for more information.
StringLib	A library that adds a number of VB string functions to NS Basic. See Tech Note 15 for more information.
SystemLib	This library adds many system calls into NS Basic, in the areas of Alarms, Files, Events, Field and Forms, HotSync Data, Localization, Preferences, Progress Manager, and System Time. See Tech Note 14 for complete detals.

12. Maximum size of Variables

Simple Variables

All numeric variables are represented internally by 8 byte floating point variables. String variables have a maximum of 32767 characters.

Arrays

In both string and numeric variables, the limit is determined by the number of elements.

```
elements=maxsub1*maxsub2*maxsub3*...
```

The maximum size of any variable is 64k.

String Arrays

The limit is defined by this formula:

```
elements*2 + (length+1 of all assigned strings) + 4
```

Array elements that do not have any value assigned yet do not take up space. If each element as just 1 character, the maximum would depend on the number of elements.

For example, DIM s(16000) requires 32000 bytes overhead. This leaves approx 32k for string values, so 32000 / 2 ("x" + null) = 16000 before the limit is reached. In the actual tests, the value is 15995, due to some overhead.

On the other hand, if "xxxx" is assigned to every element, the remaining 32k / 5 = 6400 before the limit is exceeded.

Numeric Arrays

The limit is 64k / 8 = 7999.

13. Using the Grid Control

The Grid control provides an easy way to display your data in a formatted table. It is made up of horizontal rows and vertical columns. The intersection of a row and a column is called a cell. Have a look at the sample "Grid.prj" in the Samples project to see how this control works. In the description below, the words in bold below are properties and methods of the grid control. To use them, preface them by the name of your control followed by a period.

A. Setting up the Grid in the IDE

To place a Grid on a form in your project, select the Grid object and click in the Design Screen at the top right hand corner where you want it to be placed. Set the number of **Cols** and **Visible Rows** you wish to have. Changing the number of **Visible Rows** will increase the **Height** of the grid with rows the same size. Change the **Height** of the Grid to change the size of each row. The columns are all equally spaced at Grid creation time: you can change the width at runtime. You cannot have the grid automatically hidden when the form is loaded.

If you choose the **Has Scrollbar** option, make sure the **Width** leaves room for the scrollbar to appear to the right of the object. Allow about 10 pixels for this.

B. Initializing the Grid at Runtime

You can complete the setup of your grid at runtime in the Form After code. It's a good idea to start by doing a **Hide**, so the updates you do to the table do not constantly update the screen. You can then set the column widths, types and initial values for the grid before doing a **Show**.

Set the column widths by using the **ColWidth**(*colNo*)=*p* function, where *colNo* is the column number and *p* is the number of pixels the column should be wide. Columns can hold text, number or check boxes.

To set the type of each column, use the **ColType** *colNo*, *type*, *formatString* call. Type can be "text","numeric" or "checkbox".

If the *type* of a column is "text", you can put string values into it. The *formatString* is used for the initial value of any new rows that you **Add**. It is normally "", an empty string. "text" is the default column type. Data is left justified in a text column.

If the *type* of a column is "numeric", the *formatString* defines how numbers are shown in it, using the same conventions as the FORMAT statement. For example, a *formatString* of "nnn" will display a 3 digit right justified number. The default value of a new row is 0.

If the *type* of a column is "checkbox", the *formatString* is used for an optional string value that is shown to the right of the checkbox. The default value is "", an empty string. For checkboxes without any text, a **colWidth** is 12 works well. String or numeric values can be assigned to a checkbox column cell. If the value assigned is 0, the checkbox is unchecked; otherwise it is checked. The default is that a column is unchecked.

Example
'set up a grid with 3 columns: text, numeric and checkbox

```
grid1.hide
grid1.colwidth(1)=121
grid1.ColType 1,"text",""
grid1.colwidth(2)=14
grid1.ColType 2,"numeric","nnn"
grid1.colWidth(3)=12
grid1.ColType 3,"checkbox",""
displayGrid 'call another function to set initial values
grid1.show
```

C. Populating a Grid with your own data

Grids can be populated with your own data or from a file. You cannot do both at the same time.

To populate the cells with your own data, use the **Text**, **Value**, **TextMatrix** and **ValueMatrix** functions. **Text** and **Value** set string and numeric values respectively to the current cell. The current cell is defined as the cell that is at row **Row** and column **Col**. You can change the current cell by assigning new values to the grid's **Row** and **Col** properties.

A quicker way to do this is to use the **TextMatrix** and **ValueMatrix** functions. These take the row and column values as arguments and do not require you to set the **Row** and **Col** properties separately. The **Redraw** method can then be used after all cells are updated.

To add a row, use the **Add** method. The **RowData** property can be used to store a unique value for each row, that does not get displayed. To get rid of all the rows in a grid, use the **Clear** method. The value of **Rows** will then be 0. You can also add or delete rows to a grid by changing the value of **Rows**. To get rid of one particular row, use the **Remove** method.

Example

```
MyGrid.Clear
For r=1 to 10
   MyGrid.add
   Grid.TextMatrix(r,1)="some data"
   Grid.ValueMatrix(r,2)=2
   Grid.ValueMatrix(r,3)=1
Next
```

D. Populating a Grid from a File

You can also populate your grid with information from a file. To do this, you'll need to do a bit of preparation. First, you will need to DIM a file variable to identify which file you will be loading from. Use the DIM var AS DATABASE statement. Here's an example:

```
Dim blueKey as Integer
Type dbBluesLayout
    Name as String
    age as Integer
    active as Integer
End Type
Dim dbBlues1 as Database "Blues", DbBluesRec, dbBluesLayout, BlueKey
```

We want to read in data from a file on our device called "Blues". The file variable (we will refer to the file using this) is called dbBlues1. Each record in the file has the field in dbBluesLayout. When we read a record in, it will be put into the variable dbBluesRec. The key to the file Is blueKey, an integer. This statement should be done in the same routine that you do your **BindToDatabase** call, since dbBluesRec is automatically dimensioned as a regular variable (not a global).

Now that we have defined our file, we can copy the information directly into our grid using the BindToDatabase call

```
Grid1.bindToDatabase(dbBlues1, dbBluesRec.name, dbBluesRec.Age, _
 dbBluesRec.active) Where dbBluesRec.age>=70
```

This statement copies data from the file referred to by dbBlues1 into our grid, Grid1. The next three arguments list the fields in the record which go into each of the columns. The columns do not need to be in the same order as the fields in the record layout (dbBluesLayout), nor do you need to use all the fields. The optional WHERE clause allows you to select which records to copy to the grid. You may use any expression that you could put into a normal IF statement.

If there are more records in the file selected than **Visible Rows** and you have **Has ScrollBar** set, a scrollbar will appear allowing you to see all the rows. Keep in mind that if you have a large number of rows, the scroll arrows do not work very precisely. Records are only copied into the grid as they are displayed, so there is no speed penalty for displaying large files.

As the records are read into each row, the **rowData** value of each row is set to the record number in the file. This is useful if you want to recall a record for a selected row in the grid:

```
Dim recNo as integer
Dim err as integer
recNo=Grid1.rowData(Grid1.Row) 'get rowData for the selected row
err=dbPosition(dbBlues, recNo)
err=dbRead(dbBlues, dbBluesLayout)
```

E. Interacting with a Grid

When a user taps on a grid, the code for the grid object is executed. Data cannot be typed directly into a cell. There are several useful variables that can be checked in that routine to determine what to do. The **Row** and **Col** properties will have the current row and column. You can use **Row** to get the **RowData** value to get back to the original record (if it is a bound grid) or to access other data.

You can modify the values in a grid using Text, Value, TextMatrix and RowMatrix. However, changing these values will not update the file automatically in a grid that is bound to a file. To update the file, use the rowData property to locate the file record, read it in, modify it and write it out again using your own code.

TopRow is a useful value when dealing with grids which scroll. It gives the row number of the top row that is currently displayed. You can also force a grid to scroll by changing the value of **TopRow**.

14. Modal Forms

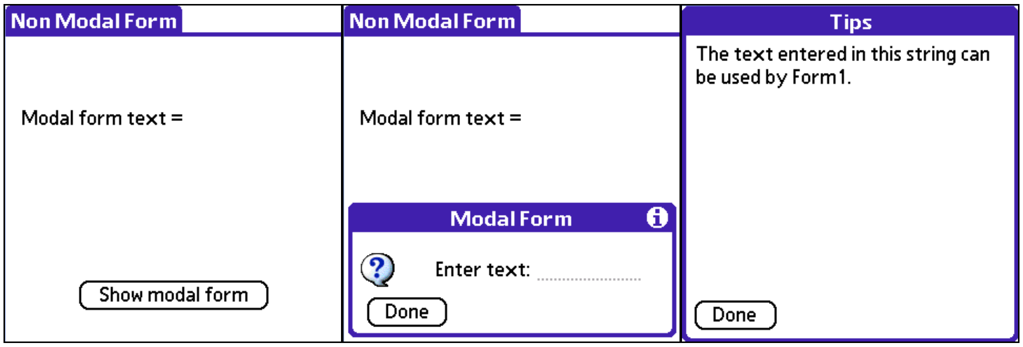
A Modal Form is a sub form that is used for a specific purpose. Modal forms are displayed on top of the current form. Modal forms have a different title bar style than modeless forms, and they have a border. Modal forms should be far less common in your application than modeless forms. Their main use is for setting options, such as application preferences. The title bar cannot be clicked on except on the "i" icon, which bring up some tips.

Modal forms do not have to be full screen - in fact, they should be no taller than necessary. Since they have a border, they look best if they are positioned 4 pixels from the left with a width of 154. They are usually positioned at the bottom of a screen. There is usually a Done button to close the form.

To make a form modal, set its Modal property to True. Enter whatever tips you want to show in the Tips property.

See the ModalForms.prj sample for more.

These screen shots show a Non Modal form, a Non Modal form with a Modal form on it and a Tips screen that is the result of tapping on the 'i" in the Modal form.



The text entered in this string can be used by Form1.

15. Navigation

Note: The Navigation features were not fully implemented in the initial release of NS Basic/Symbian OS, but were expected soon after. Check the ReadMe for more information.

Using the 5 way control, a user can tab from field to field on a form. NS Basic/Symbian OS allows you to control how this tabbing works.

Navigation is controlled by properties of the form and of objects that support navigation. Don't worry about a program with navigation defined on devices without navigation: the navigation info is simply ignored.

If you do not set up any Navigation information, it will still work. The default tab order is used, where the left and right five way buttons move through the objects in the order that they are created. You can change this order by right clicking on the form name in the Project Explorer. Up to 18 forms can have navigation set up for them in an app.

Interaction Mode vs. Navigation Mode

More complex objects need to "take over" the scroll keys in order to interact with them. For example, a text field needs to allocate all four scroll keys to move the cursor. Similarly, a pop-up list needs to use up and down to change the selection in the list. This conflicts with the requirement of using the scroll keys to navigate between objects. As a result, these complex objects need to have an interaction mode, where they can take over control of the keys. The opposite of interaction mode is navigation mode, where scroll keys navigate between objects. On a system with one-handed navigation, pressing "center" toggles between interaction mode and navigation mode. Finally, a subset of interaction mode is edit mode, which refers specifically to text fields.

Object Focus Mode vs. Application Focus Mode

Interaction mode and Navigation mode has are defined as follows. Application focus mode refers to applications that do not have keyboard navigation enabled. In this state, up and down act as page up and page down in the traditional method that Symbian OS implemented in its original form. Object focus mode refers to the state where individual objects on the screen can receive focus, essentially what "navigation mode" refers to above. Applications may or may not be able to toggle between application focus and object focus modes.

Form Navigation Properties

Nav Bottom ID: For platforms that cycle vertically, this property specifies which object receives the focus when navigating up from an object in the top row of the form (an object whose Nav Above ID is 0). A Nav Bottom ID value of zero means that focus does not cycle vertically in the form.

Nav First ID: ID of the object where focus is positioned when the form is initialized. If Nav First ID is 0, the operating system places the initial focus on the first action button, if there is one, or on the first object in the tab order if there is not.

Nav Flags:
0: (Default) The Symbian OS will decide whether to start in object or application focus mode.
1: The form will initially be in object focus mode.
2: The form will initially be in application focus mode.

Nav Jump To ID: ID of the object to which focus can "jump", if the device supports this feature. Devices can optionally have an action to trigger the movement of the focus to a commonly used object.

Object Navigation Properties

Nav Above ID: ID of the object that is above the current object. Should be 0 if the object is in the top row of the form. If the user navigates up from an object with 0 for its above object, some platforms will move the focus to the object specified by Nav Bottom ID in the form properties.

Nav Below ID: ID of the object that is below the current object. Should be 0 if the object is in the bottom row of the form. If the user navigates down from an object with 0 for its below object, some platforms will move the focus to the first object in the tab order.

Nav Flags:
0: (Default) Nothing special to do with this field.
1: If this flag is set, the object is skipped when focus moves from object to object. The object is included in the order only when it explicitly gets focus (which primarily occurs when a field, table with a field, popup trigger, or selector trigger gets focus by being tapped with the pen).
2: Used with multi-line text fields, if this flag is set the field is put in interaction mode when it receives the focus. Otherwise, the field is drawn in the focused, non-interaction mode state when it receives the focus.
Additional Notes on Navigation

Grafitti Shift Indicator and Bitmap cannot be navigated to. Label and Scrollbar are always skipped. If you navigate to slider, the arrows on the five way control the slider and do not tab.

See the NavDemo.prj sample for more information.

Index

USER COMMENT

Please use this to identify publication errors or to request changes in publications. Please let us know if you would like a reply. Return to:

NS BASIC Corporation
71 Hill Crescent
Toronto, Canada M1M 1J3
fax (416) 264-5888

An alternative method of contacting NS Basic about publication errors and requested changes is to send email to support@nsbasic.com.

The message subject should be the title of the publication followed by the printing date (from the first page).

Page	Comments