High Performance Applet Engine for Symbian OS.

For Series 60, Series 80, 7710 and UIQ compatible devices.

Developer Manual

F(x) Version 2.0

BETA 6 RELEASE - 2005-07-05
This document is under development.

http://www.symbianfx.com

Table of Contents

1 Overview of F(x)

F(x) is a real-time applet engine that utilizes C-like programming language syntax for applet creation. Applets are compiled on the fly as you edit them, providing immediate results and feedback. F(x) has been designed from the ground up for the Symbian Operating System.

Originally, F(x) has been conceived to be a powerful programmable formula evaluation engine. Over time, with development of various user interface features and programming language enhancements, F(x) has matured into an applet engine – a system that allows creation of miniature programs.

F(x) allows creation of custom applets that can be executed in the same manner on all Symbian OS user interface platforms. Because of on-device compilation and execution of applets it is possible to easily customize existing or create new applets.

The goal of F(x) software solution is to provide a versatile and customizable applet development environment that will allow you to craft your Symbian device to your personal and professional needs.

2 Document File Format

F(x) documents are stored as separate files on the file system organized in folders. F(x) allows for a file-manager-like navigation to select and manage desired documents. F(x) documents are stored using UTF-8 file format compatible with traditional ASCII. F(x) also understands Big-Endian Unicode encoding. Any file encountered in Unicode format will be automatically converted to UTF-8 when saved.

F(x) document file format allows document creation and editing using MS Windows Notepad application available on any Microsoft Windows OS. Unicode storage of documents allows them to contain text in any language.

Example of Japanese language (Unicode character set) used within F(x) applets.

3 File Management

File management view of F(x) resembles a traditional file manager view, where you have a list of files and folders to categorize them. Selecting a folder with a stylus or navigational keys will enter the folder. Selecting a file with a stylus or navigational keys will open it for editing.

File Management on UIQ Platform

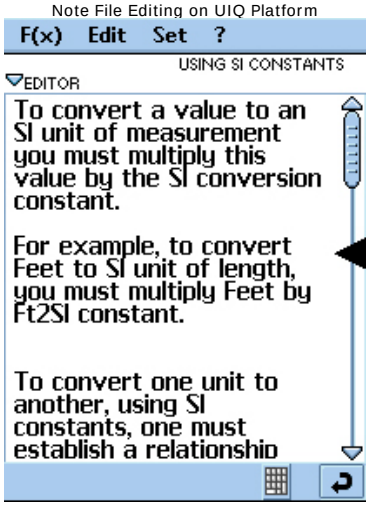
Once editing of the file is complete, you can exit back to the main file list screen by tapping the “Go Back” button at the bottom right of the screen. At this point, the file will be saved.

Various file management operations are available through the FX application menu. These operations include creation of new folders and files (applets, notes and constant definition files), renaming/relocation of the existing files etc. Some of these operations can be performed on multiple files simultaneously. To select multiple files simultaneously, files should be tapped with a stylus on their icon to the left of the file name. This operation will select the file but will not open it for editing. (Selection of multiple files is available only on Symbian platforms that support a pointing device.)

4 Note Files

Notes are text files provided as a convenient way for applet documentation. Creating and editing a note is similar to creating and editing an applet. To view or edit a note, simply click on the note file (note file is distinguished by the “notepad” icon on the left side of the file list). To create a note file choose “F(x)/New Note” menu option in the file manager.

Note File Editing on UIQ Platform

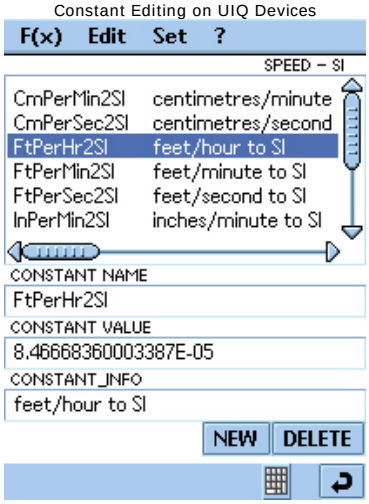


5 Constant Definition Files

Constant definition files contain lists of constants that can be used within F(x) applets. For each constant definition file, F(x) creates a constant category with the name of the file. This provides an easy way to group various constant into categories. Constant definition files are distinguished by the "C" icon on the left of the file management view. To add/remove constants in a file, select it in the file manager. To create a new constant definition file choose "F(x)/New Constant Set" menu option when in file manager.

Please note that removing constants from existing constant definition files or deleting constant definition files can cause certain applets to stop working since they may be relying on the deleted constants to perform their calculations.

Constant Editing on UIQ Devices



6 Table (CSV) Files

F(x) allows storage and retrieval of data from Tables. Tables are traditional CSV (Comma Separated Value) files that always reside on the file system in their original text format.

CSV files are compatible with any mainstream spreadsheet applications such as Microsoft Excel, Lotus 1-2-3 and Open Office.

F(x) applets can retrieve data from and save data to CSV files. These files can also be opened in the text editor and modified manually. Contents of CSV files can be transmitted via E-Mail, SMS or MMS.

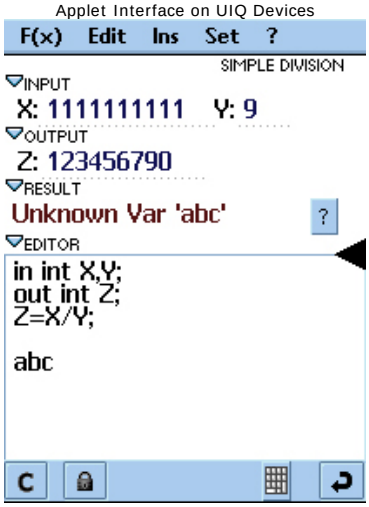
Table access functions also offer a convenient way for persistent storage of information as well as a light database-like functionality.

7 Applet Files

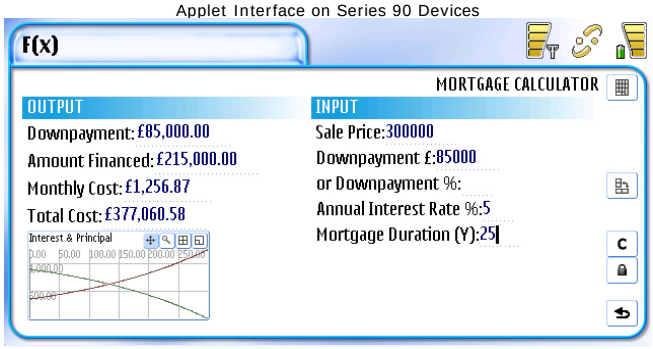
Applet files are text files that are intended to contain implementation of applets. When editing an applet, F(x) tracks modifications to it and recompiles/reinterprets the applet on the fly providing you with the real-time compiler feedback and applet execution.

To view, edit or execute an applet, simply click on the applet file when in the file manager (applet file is distinguished by the "F(x)" icon on the left of the file management view).

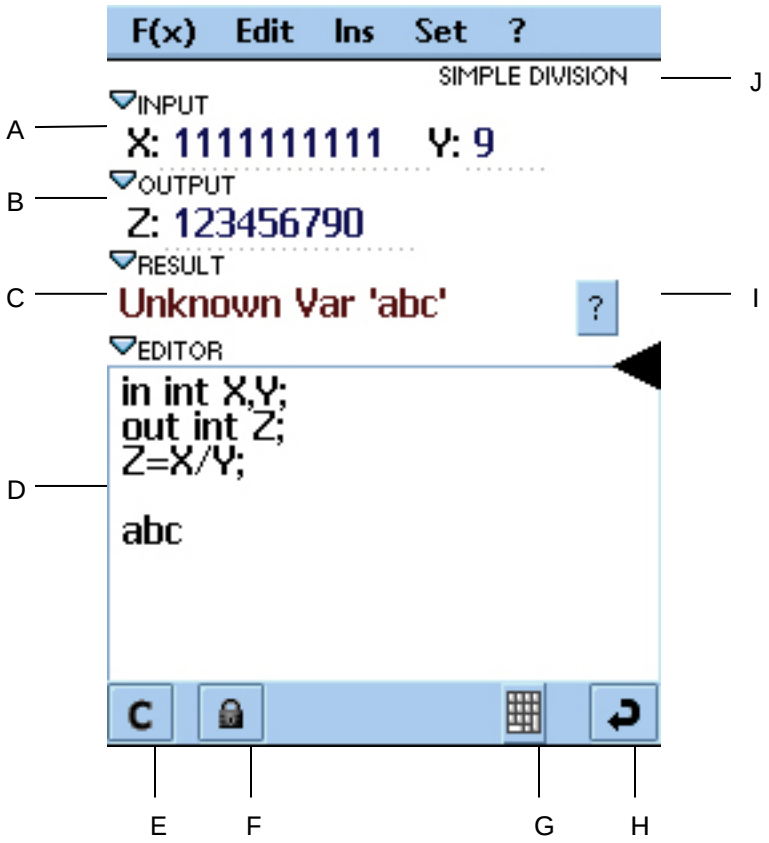
Applet Interface on UIQ Devices



Applet Interface on Series 90 Devices



8 Applet Editing Interface



A. Dynamic Variable Input
B. Dynamic Variable Output
C. Applet Result or Error
D. Applet Editor
E. Dynamic Variable Input Reset (Clear Inputs menu on Series 60 platform)
F. Dynamic Variable Interface Lock (View menu on Series 60 platform)
G. Keypad Accelerator
H. Go Back
I. Error Locator (Positions cursor at the error location)
J. Applet Name

Please refer to Section 13 - Dynamic Variables for information on dynamic variables.

9 Using Data Input & Output User Interface

Switching between dynamically created controls can be done via keyboard by using UP/DOWN arrow keys on your device (or UP/DOWN scroll wheel on Sony Ericsson UIQ devices).

While navigating and selecting controls using a pointing device is a straightforward process, devices running Series 60 and Series 80 user interface platforms require key interaction.

Following controls are capable of handling keys:

Slider

“Right Key”	-	Increment the slider value by step.
“Left Key”	-	Decrement the slider value by step.

Enum (Multiple Choice Selection) or Unit Selection

“Select Key”	-	Bring up the list of items for selection.

(Select is typically a push down key on the center of the joystick control)

Graphs and Charts

“0”	-	maximize or restore to the original default position.

Graphs

“Select”	-	Enter/exit the graph interaction mode.
“1”	-	Switch to move mode.
“2”	-	Switch to zoom mode.
“3”	-	Reset pan and zoom to default.
Joystick	-	Modify pan or zoom settings.

When in the interaction mode, the graph is highlighted in red color and navigating using joystick allows you to pan and zoom on the contents of the graph.

Array

“Select”	-	Enter/exit the array interaction mode.
Joystick	-	Pan

Checkbox, Button

“Select”	-	Check / Uncheck the checkbox or press on the button control.

10 F(x) language fundamentals

F(x) syntax is closely related to the C language syntax. F(x) utilizes a "relaxed language parser" which will at times ignore invalid syntax such as extra commas and will signal errors only when it no longer is able to interpret the applet syntax correctly.

10.1 Statements

Statement is a single operation that is separated by a semicolon. A statement block can be created by enclosing multiple statements within "**{**" and "**}**" brackets. A statement block is typically treated as a single statement.

10.2 Expressions

Expression is a single statement that produces a single result.

10.3 Comments

F(x) supports C++ style single line comments that are prefixed by '//' symbols. Once F(x) encounters '//' symbols in the applet, it will consider the rest of the line as a comment.

Please note, that due to a limited screen size of mobile devices, the text line might be wrapped by the editor, making it look like the comment continues on the next line.

Example:

```
// This is a comment
```

Comments can also be inserted into the applet following the "eof" keyword (please refer to Section 19 - Keywords for additional information.

11 Variable Declaration

Syntax:

<variable type> <variable name> [= <expression>];

Example:

```
var ItemCount;
real mass = 3.5 / 12;
```

F(x) supports use of unlimited variables. Variable names can not match the names of internal functions or constants. Variable names are not case sensitive.

12 Variable Types

F(x) supports a number of variable types that can be used in declaration of variables and will determine how variables are treated during calculations as well as how they will be displayed.

12.1 Supported Variable Types

var	- variant type (assumes real, reserved for future use)
int	- integer with decimal representation
real	- floating point representation
float	- alias for real
double	- alias for real
bool	- boolean (true/false)
byte	- 8 bit calculations (clamps values to 0..255)
char	- character symbol (0..0xFF) representation
unicode	– unicode symbol (0..0xFFFF) representation
hex	- integer with hexadecimal representation
dec	- integer with decimal representation (same as int)
oct	- integer with octadecimal representation
bin	- integer with binary representation
funds	- fixed point currency representation according to current device local settings (intended for financial calculations).
string	- text string
date	- integer timestamp (same as int)
complex	- complex number

Variable type indicates how variable values will be treated throughout applet execution and displayed in the applet result (or dynamic variable output). Please see “Numerical Representations” section for additional information.

Some variable types are redundant (like real and float). This is done to simplify applet creation for people who already have experience with programming and are used to certain data types.

12.2 Variable Type Interpretation

F(x) performs calculations using two internal data types: 64 bit floating point and 64 bit signed integer. When operating on integer types, all calculations are integer. When operating on floating point types or when mixing integer and floating point calculations, all calculations are floating point. To perform integer and binary arithmetic operations on floating point types, F(x) will convert values to integer, perform required operations and then convert the result back to floating point.

12.3 Silent Type Conversion

F(x) tries to perform silent type conversion when mixing numerical types and strings. If a string is assigned to a numeric type, it reads the contents of the string as a value of that numerical type. If a number being assigned to a string, it is converted to a string representation.

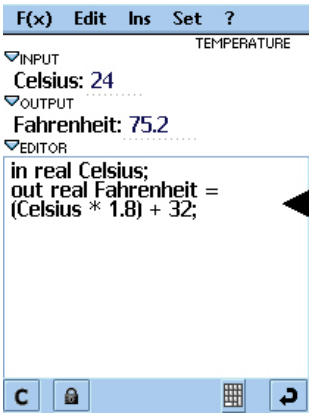
If adding strings and numbers, the result will be the same type as the first element in the addition. For example: 123+”456” results in 579 while “123”+456 results in “123456”.

Please note that only decimal numerical representation is supported when converting data to/from string.

13 Dynamic Variables

Dynamic Variables offer user to input or output default variable value without having to modify the applet source code. Declaration of a dynamic variable creates an interface at the top of the application that allows the user to specify the variable value. The default input/output interface for dynamic variables is an edit prompt capable of expression evaluation. Other interfaces including sliders, check boxes and choice selection popups are also available.

13.1 Declaration



Syntax:

in *<variable type> <variable name>;*
out *<variable type> <variable name> [= <expression>];*

Example:

in real *Celsius;*
out real *Fahrenheit = (Celsius * 1.8) + 32;*

13.2 User Input

Dynamic input variable edit prompt understands simplified mathematical expressions.

Example:

*sin(0.5)*5634/(34+67);*

13.3 Content Reset Button

The dynamic variable content reset button located at the bottom left of the applet editing interface clears all the user entered content in the dynamic variable input fields.

13.4 Interface Lock Button

Interface lock button cycles between the following user interface configuration modes:

- Variable input / output only.
- Variable input / output and the applet editor.
- Applet editor only.

On Series 60 this functionality is available via the “View” menu when the applet is opened.

Interface lock can be used to prevent user interface from periodically reconfiguring itself when editing a formula that contains dynamic variables.

13.5 Landscape / Portrait Mode Button

Landscape / Portrait Mode Button (available on Series 90 devices only) switches the layout of the input and output user interface areas by placing them either one on top of another (as done in the UIQ and Series 60 layouts) or side by side, allowing for more vertical space. Landscape mode is available on Series 80 (via the application menu) and on Series 90 devices only.

14 Dynamic Variable Interfaces

14.1 Declaration Syntax

When declaring dynamic input/output variables, you can change their properties such as their name as it appears in the input/output prompts or the interface they use to input/output data.

14.2 Naming

When declaring dynamic input/output variables, you can change variable name to appear differently in the input/output prompts. To do this, you have to use a *variable modifier* as follows:

Syntax:

@<variable name> "<variable title>";

Example:

```
in real Fo;
...
@Fo "Objective Focal Length"
```

This example changes the input prompt:

"Fo: ....."

to:

"Objective Focal Length: ....."

It is a good practice to place variable modifiers at the end of the applet so that they do not interfere with readability of the applet source code.

14.3 Default Expression Editor

When no variable interface has been specified, the **EDIT** interface is used as a default. This interface offers expression input.

Example:

```
in float f;
@f "Try Expression Here";
Or
@f edit "Try Expression Here";
---
When entering values you can use expressions:
Try Expression Here: 23+45.6+77.8/3
```

EDIT interface also allows entry of data into an array:

Example:

```
in float f[];
@f “Enter Array”;
---
Enter Array: [1 2 3];
```

14.4 String Input/Output

When using EDIT interface to enter data, in order to enter text strings, they have to be enclosed in quotes. To avoid having to do this, STRING interface is available:

Example:

```
in str name;
@name string “Client Name:”
---
Client Name: Jane Doe
```

14.5 Unit Input/Output

If dynamic variable is used to input/output SI compatible unit, it can be specified as a “unit variable”. In addition to the value input field, unit variables display a unit selector that can be used to specify the units in which the value has been entered.

Unit variable interface declaration syntax is as follows:

***@**var **unit** <category> <default unit type> [“<variable title>”];*

@var identification of the variable to which this interface should be attached.

<category> is the SI constant category

<default unit type> is the SI unit type that belongs to the specified category. The value entered by the user will be converted to this unit when assigned to the variable.

Example:

```
float W,P;
@W unit weight kg “Weight”;
@P unit pressure kPa “Pressure”;
```

Please note that Unit Input interface supports expression input.

14.6 Enum Input (Value List)

Enum variable interface offers the user to make a single value selection from a list of pre-defined items.

Syntax:

*@var **enum** <array> [<default value>] [“<variable title>”];*

@var identification of the variable to which this interface should be attached.

<array> - list of items to be presented to the user

<default value> - default value to be selected (must be a value present in the array). If not specified, the first element of the array will be used as the default selection.

Example:

```
float L,Sex,P;

@L enum [1 2 3] 2 “Level”;
@Sex enum [“Male”, “Female”];
@P enum [“YES”, “NO”] “YES” “Present”;
```

14.7 Other Interfaces

F(x) offers different variable interfaces for input and output of data. Please consult Functions and Interfaces document that comes with F(x) for more information and interface parameter specifications.

15 Static Variables

Static variables are variables that are persistent between applet executions. Variables in F(x) are created at the beginning of the applet execution. Static variables, however, are preserved between applet executions and are reset only when the applet is being recompiled.

The following Fibonacci applet does it's processing using a static variable:

```
in int x;
out int y;

static int a[];  // declares a static (persistent) array variable
if (isinit())      // check if applet has been just compiled (first run)
  a = [1 1];    // if yes, initialize the static variable
int c = count(a);       // count elements in the array
if (c < x)
{
  a = append(a, a[c-1]+a[c-2]);     // append data to the static array
  respawn();   // tell the system to run the applet again
}
else
  y = a[x-1];  // finished, obtain the resulting value
```

Please note, applet is compiled when it is being loaded or when it is being edited in the source code editor. When user enters data in the dynamic variable interfaces, the applet is executed but not compiled.

Example:

```
in int i;
static int result;
result += i;
```

This example will create an entry prompt for the variable i. When the user enters numbers 1, 2 and 3 (resulting in 123) the static result of the applet will be 136 because on each number entered the applet will be rerun without resetting the static variable. 136 = 1+12+123.

16 Numerical Representations

16.1 Input Format

Depending on the format in which the number is entered, F(x) will interpret the number in a different numerical representation. The format in which number is entered is determined by a prefix or a suffix to the number.

Integer Decimal Representation (numbers only)

123

Floating Point Decimal Representation

123.123 123e-3 123e+10 123e10

Hexadecimal Representation ('0x' prefix)

0xABCD

Octadecimal Representation ('0c' prefix or 'c' suffix)

0c1234 1234c

Binary Representation ('0b' prefix or 'b' suffix)

0b0101 0101b

16.2 Output Format

The display format of a variable is determined by its type. For example, '**hex**' type will cause variable to display as 'ABCD', while '**bin**' type will cause it to be displayed as '0101';

'**funds**' type should be used for currency calculation as variables of this type are formatted as currency in accordance to the current device locale settings (Accessible via Control Panel). If locale settings have been modified while F(x) is running, you will need to restart your Symbian device for changes to take effect.

16.3 Result Output Format

Display format of the result can be overridden via the "Result" menu.

17 Escape Sequences

Escape sequence is a special sequence of characters used for describing non-printing characters. Escape sequences start with a backslash and are followed by the control character.

Following escape sequences are supported by F(x):

\n	Newline
\t	Horizontal tab
\\	Backslash
\'	Single quote
\"	Double quotes
\x	Single byte hexadecimal number
\u	Double byte hexadecimal number

Example:

"Line one\nLine two\n\"Line three is enclosed in quotes\""

Escape sequences can be used to enter raw hexadecimal values in a string. Two methods are available for entering hexadecimal values, single byte and double byte values.

All internal data processing in F(x) is performed using UTF-16 (double-byte Unicode). The '\xNN' escape sequence allows entry of single byte hexadecimal numbers where each number is considered to be maximum of 0xFF value.

Example:

*"**\x41424344**"*
*is interpreted as **0x41 0x42 0x43 0x44** and results in "**ABCD**"*

The "\uNNNN" escape sequence allows entry of double byte hexadecimal numbers where each number is considered to be the maximum of 0xFFFF value.

Example:

*"**\x0041004200430044**"*
*is interpreted as **0x0041 0x0042 0x0043 0x0044** resulting in "**ABC**"*

Hexadecimal escape sequence is terminated when a non-hexadecimal character is encountered.

For additional information and Unicode character tables please consult http://www.unicode.org.

18 Operators

18.1 Unary

-	-	Unary Minus	- <expression>

18.2 Arithmetic / Algebraic

=	-	Assignment	<variable> = <expression>
+	-	Addition	<expression> + <expression>
-	-	Subtraction	<expression> - <expression>
*	-	Multiplication	<expression> * <expression>
.*	-	Per-Element Multiplication	<expression> .* <expression>
/	-	Division	<expression> / <expression>
./	-	Per-Element Division	<expression> / <expression>
%	-	Integer Modulo[1]	<expression> % <expression>
+=	-	Add and Assign	<variable> += <expression>
-=	-	Subtract and Assign	<variable> -= <expression>
*=	-	Multiply and Assign	<variable> *= <expression>
.*=	-	Per-Element Multiply and Assign	<variable> *= <expression>
/=	-	Divide and Assign	<variable> /= <expression>
./=	-	Per-Element Divide and Assign	<variable> /= <expression>
%=	-	Assign Modulo (int)	<variable> %= <expression>
**	-	Power	<expression> ** <expression>
**=	-	Assign Power	<variable> **= <expression>
++	-	Increment	<variable> ++
--	-	Decrement	<variable> --

18.3 Conditional

==	-	Equal	<expression> == <expression>
!=	-	Not Equal	<expression> != <expression>
<	-	Less	<expression> < <expression>
<=	-	Less or Equal	<expression> <= <expression>
>	-	More	<expression> > <expression>
>=	-	More or Equal	<expression> >= <expression>
? :	-	Conditional expression: <conditional expression> ? <expression A> : <expression B> If <conditional expression> is true, <expression A> will be executed, otherwise <expression B> will be executed.	

18.4 Logical

&&	-	Logical And	<expression> && <expression>
||	-	Logical Or	<expression> || <expression>
!	-	Logical Not	!<expression> Expression is pre-converted to Boolean (integer 0 or 1).

[1] % operator is used to perform an integer only modulo operation. The expression is converted to integer before being processed. Please refer to the fmod() function to obtain floating point remainder.

18.5 Binary

<<	-	Left Shift by N bits	<expression> << <# bits>
<<=	-	Assign Left Shift by N bits	<variable> <<= <# bits>
>>	-	Right Shift by N bits	<expression> >> <# bits>
>>=	-	Assign Right Shift by N bits	<variable> >>= <# bits>
&	-	Binary And	<expression> & <expression>
&=	-	Assign Binary And	<expression> & <expression>
|	-	Binary Or	<expression> | <expression>
|=	-	Assign Binary Or	<expression> | <expression>
^	-	Xor	<expression> ^ <expression>
^=	-	Assign Xor	<expression> ^ <expression>
~	-	Binary Not (Invert)	<expression> ~ <expression>

18.6 Other

[ ]	-	Subscript Operator (Array declaration or dereference).
‘	-	Matrix Transpose

18.7 Operator Precedence

The operators at the top of the list are evaluated first (in the order listed):

Precedence	Operators
1	( ) [ ]
2	! ~ ++ -- ‘
3	–(unary minus)
4	** (power of)
5	* / %
6	+ -
7	<< >>
8	< <= > >=
9	== !=
10	& (binary AND)
11	^ (XOR)
12	|
13	&&
14	||
15	? :
16	=

19 Keywords

if, else Please see Section 25 - Conditions

in, out Please see Section 13 - Dynamic Variables

for, while, do while
Please see Section 26 - Loops

halt Terminates applet execution at run-time. The interpreter will exit keeping the last calculated result as the applet result.

Example:

```
if(a < 100) halt;
a – 100;            // continue evaluation if not halted
```

eof "end of applet" - terminates applet compilation. The compiler will compile the applet only up to the point it encounters this keyword. Any text that follows this keyword will be ignored.

20 Execution control functions

F(x) includes functions that allow conditional termination of the applet execution or signaling of an error:

error(<id>)

This function will terminate applet execution and report the error to the user. Possible error values are defined as system constants as follows:

E_INPUT	-	Invalid User Input
E_RANGE	-	Out of Range
E_INSUFF	-	Insufficient Input
E_PARAMS	-	Too Many Parameters

error("<string>")

Thus function will terminate applet execution and report the error to the user. The error string to be reported is specified as the parameter to this function.

range(<value>,<min>,<max)

This function will terminate applet execution with the "Out of Range" error if the supplied value is outside of the supplied minimum and maximum boundaries.

21 Arrays, Vectors and Matrices

21.1 In-place Arrays Declarations

In-place arrays are temporary arrays constructed within a statement and can be used as a part of an expression or to initialize a variable.

One dimensional in-place arrays can be declared using the following syntax:

[1 2 3] or [1, 2, 3]

Two dimensional in-place arrays can be declared using the following syntax:

[[1 2 3] [4 5 6]] or [1 2 3; 4 5 6]

Above example creates the following two dimensional array:

1 2 3
4 5 6

Each element of the in-place array declaration can be an expression:

[sin(x), 0.0; 0.0, cos(y)]

Above example creates a 2x2 array where [0][0] element is “sin(x)” and [1][1] element is “cos(y)”.

Please note that mixing numerical and string elements in the in-place array declaration will cause all elements to be converted to string. For example, declaring [1 2 3 “a”] will result in [“1” “2” “3” “a”].

21.2 Array Padding

All rows in a multidimensional array must be of the same length. If array is created with different row sizes, all rows will be padded with zeroes to match as follows:

[1 2; 4; 7 8 9]

will result in the following two dimensional array:

1 2 0
4 0 0
7 8 9

21.3 Comma vs. Space Array Element Separation

PLEASE NOTE: While it is possible to use either space or comma when initializing an in-place array, a minus preceding a negative number may be considered as a part of an expression as follows:

[1 -5 3] will be treated as [(1 -5) 3] and create array [-4 3]

Thus a comma must be used to separate these values as follows:

[1, -5, 3] will create [1 -5 3]

21.4 Zero Size Array Variables

Array variables in F(x) are dynamically resized to accommodate data that is being placed in them. This means that at declaration a variable can be an array of zero size.

The following syntax declares one, two or three dimensional array variables of zero size:

```
float a[];
float a[][];
float a[][][];
```

No operations on these variables can be performed other then assigning another array that contains data. An array variable can be assigned only an array that contains same number of dimensions.

21.5 Pre-allocated Array Variables

Array variable can be pre-allocated at declaration as follows:

```
float a[3];
float a[3][3];
```

Above declarations will create a variable array that contains elements of type float inialized to 0.0.

21.6 Matrices and Per-Element Array Operations

Any operation on the array are performed on per-element basis with the exception of multiplication where an array will be treated as a matrix and follow the rules of linear algebra matrix multiplication.

In order to perform a per-element array operation, the operator must be preceded by a dot '.' as follows (in case of multiplication): '.*'

Example:

Matrix Multiplication: *[2 3; 4 5]* ***** *[6 7; 8 9] = [36 41; 64 73]*
Per-Element Array Multiplication: *[2 3; 4 5]* **.*** *[6 7; 8 9] = [12 21; 32 45]*

21.7 Matrix Multiplication

The multiplication operator '*' is executed according to the rules of linear algebra. If A and B are two matrices, the A*B operation can be carried out only if the number of columns in matrix A is equal to the number of rows in matrix B. The result is a matrix that has the same number of rows as matrix A and the same number of columns as matrix B.

Please note that matrix multiplication is not commutative (A*B != B*A).

21.8 Per-Element Array Multiplication

The per-element multiplication operator '.*' performs multiplication of each element in the array A on the corresponding element of the array B.

Multiplication of all array elements between A and B arrays is only possible if A and B are arrays of the same size.

21.9 Matrix Transpose Operator

Matrix Transpose operation interchanges row and column dimensions of a matrix. F(x) uses the ' **'** ' operator placed after a matrix to transpose a matrix.

Example:

[1 2; 3 4]' produces [1 3; 2 4]
[1 2 3]' produces [1; 2; 3]

Following examples demonstrate use of transpose operator:

```
A = B';
A = [1 2 3]';
A = (B*C)';
```

Transpose operation can be also performed on a matrix using **mtranspose()** function.

22 Complex Numbers

F(x) provides integrated support for complex numbers via the user of **complex** data type. Complex numbers can be converted from and to an array with two elements where first element of the array represents the real part of the complex number and the second element represents the imaginary part.

Example:

```
complex a = [1, 1];
a;  // the displayed result will be (1,1)
```

Complex numbers support a full set of arithmetic and logical operators.

When using complex numbers, it is possible to mix in two element arrays. If one of the elements in the expression is a complex number and another one is a compatible array, the expression will be considered as an operation on complex numbers.

Example:

```
complex a = [1, 1];
complex b = a ** [0.5, 0.0];
complex c = a + [2, 2];
```

F(x) allows use of arrays with complex numbers as well as conversion from and to arrays that may represent complex numbers as pairs of real numbers.

Example:

```
complex a[] = [1,1; 2,2; 3,3];
real b[] = a;          // b results as [1,1,2,2,3,3];
real c[][] = a;        // c results as [1,1; 2,2; 3,3];
```

F(x) offers a variety of mathematical functions that operate on complex numbers. For the list of these functions please consult Functions and Interfaces document included with F(x) installation or refer to the online documentation available at http://developer.symbianfx.com

Examples:

Finding the distance between two real vectors:

```
real d = dist([1 2 3],[4 5 6]);
```

Finding distance between two complex vectors:

```
real d = dist([1,0;2,2;3,0],[4,0;5,0;6,0]);
```

Finding the sine of a real number:

```
real s = sin(1);
```

Finding the sine of a complex number:

```
real s = sin([1,0]);
```

23 Strings

23.1 Using Strings

A string is declared in-place by enclosing text within double quotes:

“this is a string”

In-place string declarations may be up to 128 characters in length (including quotes).

Strings can be assigned to a variable of type string:

string s = “this is a string”;

Strings can be used within a string array:

string s[] = [“first” “second” “third”];

23.2 String Comparison

Two strings can be compared using ‘==’ and ‘!=’ operators. String comparison is not case sensitive.

The unary NOT operator ‘!’ allows to check for an empty string

```
str s = “”;      // create an empty string
if(!s) { … }     // tests true for an empty string
if(!strlen(s)) { … }     // alternate method to test for an empty string
```

23.3 String Concatenation

Strings can be concatenated using ‘+’ operator.

23.4 Numeric data type conversion

Strings offer silent conversion from and to numeric data types. When converting from and to strings, all numbers are considered to be floating point numbers.

```
str s = 1;       // results in s being “1.0”
real r = “1”;    // results in r being 1.0
int I = “1.0”;   // results in i being 1
```

It is possible to convert a number to a string representation of an integer by using itoa() function.

```
str s = itoa(1);        // results in s being “1”
```

itoa(), atoi(), ftoa(), atof() functions offer strict data type conversion.

When used in expressions with numeric data types, the order in which types are used dictates the result.

```
123 + “456” results in 579
“123” + 456 results in “123456”
```

Other operations

F(x) provides a set of string manipulation functions such as strlen() to obtain a string length, strops() to obtain position of a substring within a string and substr() to retrieve a substring from within a string.

24 Constants

Constants can be used in the applet itself as well as a part of the dynamic variable input expression. Constants are created as “data sets”, which are files containing constant names and values. These files are processed and cached at F(x) startup.

24.1 Duplicate constants

If F(x) detects presence of duplicate constants (constants that have the same name), F(x) will report the problem and log all relevant information into the F(x) log file “Error Log” located in the root F(x) folder.

24.2 Constant dataset file format

Constants are stored in a UNICODE file in the following format:

CONSTANT_NAME = CONSTANT_VALUE; <CONSTANT_INFO_LINE>

CONSTANT_NAME	- an alphanumeric name starting with a letter.
CONSTANT_VALUE	- a numeric value that is represented in a hexadecimal, decimal, octadecimal or binary formats (please refer to Section 16 - Numerical Representations).
CONSTANT_INFO_LINE	– an optional constant descriptor that may contain any text.

If you have your own constants that you would like to include into F(x), you can assemble them on your personal computer in the above described format, place the file in one of the F(x) folders and update caches (or restart F(x)).

24.3 Unit conversion constants

F(x) offers unit conversion constants that are not a part of the constant database, their values are determined at the compile time. Unit conversion constants are formatted in the following way:

<source unit abbreviation>2<destination unit abbreviation>

For example, to convert Kilometers to Feet we can use “Km2Ft” constant.

Unit abbreviations used in the constant library may not follow common scientific notations but rather rules established in F(x):

Square and Cubit units are denoted by “Sq” and “Cu” suffixes. For certain abbreviations force is denoted with “F” suffix. Spread Rate related constants all have “Sr” suffix. Seconds unit is always abbreviated as “Sec” and Hour unit is always abbreviated as “Hr”, thus traditional abbreviations such as “kWh” and “Kph” will become “KwHr” and “KmHr”.

In addition, abbreviations of some units may include cultural/country information, for example, “UK Ton 1016.046909Kg (long)” will be denoted as “TUK” while “US Ton 907.18474Kg (short)” will be denoted as “TUS”. A regular metric “Ton 1000Kg” is denoted as “T”.

Examples: Ft2In (Feet to Inches), Ft2M (Feet to Meters), MiSq2KmSq (Square Miles to Square Kilometers), Oz2G (Ounces to Grams).

Please refer to the Appendix B – Unit conversion for additional information.

25 Conditions

Conditional statement execution can be created by using if…else statement.

Syntax:

if*(<expression>) <statement> [* ***else*** *<statement>]*

Example:

```
if(Count < 5) Count = 5;
if(Count < 5) { Count = 5; Parts = 3; }

if(Value < 10)
{
        Value = 10;
}
else
{
        Value = 20;
}
```

Please note that conditional expressions are evaluated as regular language expressions, meaning that conditional statement if(f1(a) || f2(b) || f3(c)) will evaluate calls to f1(), f2() and f3() even if f1() return TRUE. This is different from some traditional programming languages where the condition may be considered TRUE if call to f1() satisfies it. In C/C++ programming languages if f1() returns TRUE, f2() and f3() will not be called.

26 Loops

F(x) provides traditional support for loops as in any other programming language.

Please note - Loops present possibility for potential performance loss during applet evaluation as well as performance resource drain on the host device. Given these factors, applets are currently limited to the execution of 131072 instructions maximum. If evaluator encounters more instructions, the applet will be considered as “Infinite” and an error will be reported.

Syntax:

for*([initialization statement] ; <condition>; [iteration operation]) <statement>;*

Example:

```
for(int a = 0; a < 25; a++) b += c;
for(; a < 100;)
{
        a += k;
        k *= m;
}
```

while*(<condition>) <statement>;*

Example:

```
while(int a < 100) a += k;
```

do *<statement>* ***while****(<condition>);*

Example:

```
do { a += i; i++; } while(i < 10);
```

27 Expression Results

F(x) internal instruction set is similar to that of a modern computer programming language. Similarly to other execution environments, F(x) executes its operations on a dynamic program stack. The F(x) result window always displays the result of the last operation performed on F(x) stack.

Example:

1 + 2 will produce 3 as the last value on the stack and result window will display ‘3’.

If you are performing complex operations with multiple variables and would like to place one of the variables on the stack for output in the result window, you may simply place the variable containing the value or a constant at the end of the applet.

Example:

```
if(a < 10)
{
        0;      // place 0 as the last applet result value
        halt;   // halt applet execution
}
else
{
        a += 50;
}
a;      // place variable a as the last applet result value
```

28 Value Sampling

sample() function allows sampling of a given expression for a set of values. On input this function takes an expression to be evaluated, the starting value of the evaluation, the ending value and the amount of samples. On output this function returns an array with evaluated samples.

Syntax:

sample("<expression>",<start>,<end>,<number of samples>);

Example:

*sample("sin(x)*0.5",-PI,PI,25)*

The above function call will return an array of 25 elements that will represent the sampling of the supplied expression between –PI and PI values.

The variable X is automatically created in the applet that uses sample() function. If user creates variable X in the applet, it will be used as a sampling range counter during sampling.

The outside environment of the compiler is visible to the sampled expression, thus all variables that are currently declared in the applet can be used within the specified expression:

```
in float MyValue;
out float graph[] = sample("cos(x)*MyValue",-5,5,10);
```

29 Graphing and Plotting

29.1 One and Two Dimensional Graphs

One dimensional graphs are displayed on Y over X axis where position on the Y axis is the value of the array element and position on the X axis is the sequential number of the array element.

Two dimensional graphs take a two dimensional array and consider column 0 to represent X axis and column 1 to represent Y axis.

If one dimensional graph receives a multi-dimensional array on input, it will consider each row of the array as a separate dataset and plot each dataset on top of each-other. This ability can be used to plot multiple graphs on top of each-other for comparative analysis.

Syntax:

```
@array graph [<min X> <max X> <min Y> <max Y>] [“<caption>”];
@array plot [<min X> <max X> <min Y> <max Y>] [“<caption>”];
@array stem [<min X> <max X> <min Y> <max Y>] [“<caption>”];

@2d_array graph2d [<min X> <max X> <min Y> <max Y>] [“<caption>”];
@2d_array plot2d [<min X> <max X> <min Y> <max Y>] [“<caption>”];
```

Interface Parameters:

> *<min X> <max X> <min Y> <max Y> - graph view bounding values. Default values 0,1,0,1 are used if none supplied. These values are used by the graph interface for initial plot dimensions.*

Example:

```
out float a[] = [0.4 0.5 0.8 0.3 0.6];
@a graph; // default view bounds used

out float a[] = [0.4 0.5 0.8 0.3 0.6];
@a graph 0 1 0 1; // initial view bounds supplied

// ---
// following example “fits” the graph in the view

float a[] = [0.4 0.5 0.8 0.3 0.6];
@a graph 0 count(a) min(a) max(a);

// min Y is the smallest element in the array
// max X is number of array elements
// max Y is the largest element of the array
```

30 Function Plotting

30.1 Sampling and plotting functions

As it is seen in the above examples, F(x) does not support a concept of function plotting in a way similar to calculators. Instead, it allows sampling of function values into an array and then provides a graph interface by means of which this array can be plotted.

The following example reviews function plotting in F(x):

```
// offer user to input function, for example “sin(x)/x”
in str func;

// set interface for variable “func” to string to avoid having
// user to include his input in quotes.
@func string “Function”;

// offer user to input sampling bounds (x1…x2)
in float x1,x2;

// specify sampling resolution (how many samples to perform)
float samples = 256;          // N samples

// create output array that will be plotted as a graph
out float a[];

// sample user specified function func for N samples
// with x value ranging between x1 to x2.
a = sample(func,x1,x2,samples);

// setup graph interface
@a graph x1, x2, min(a), max(a) “Function Plot”;
```

In this examples, our x coordinate bounds are x1 and x2 and our y coordinate bounds are the smallest and the largest value found in the array a.

It is important that @a graph…; interface is setup after the sampling is performed because it’s parameters min(a),max(a) rely on the data present in the array a.

30.2 Plotting bounds

Graph interface requires 4 parameters in order to determine plotting bounds. Values in the array will be plotted using absolute Y value within virtual X bounds. Meaning that for Y axis, the plotting coordinates will always be that of the absolute Y value, however for X axis, the plotting will be interpolated between x1 and x2.

Example:

```
float a[] = sample(“sin(x)”,-10,10,100);
```

```
@graph -10, 10, -10, 10;
```

This example will create a graph with X bounds ranging -10, 10 and Y bounds ranging -10, 10. Array a[] contains 100 samples of sin(x) where x ranges from -10 to 10. This array will be plotted on X axis where a[0] is located at -10 and a[99] is located at +10. However, the value of a[x] will remain absolute (not scaled), so even if the graph displays Y axis ranging from -10 to +10, the Y value will range between -1..+1 (range of the sin(x) function output).

30.3 Function variable access

Please note that function specified in call to sample() has access to variables present in the current applet environment. Thus the following scenario is possible:

```
in float t;
out float a[] = sample(“cos(x)*sqr(x*t)”,x1,x2,S);
```

30.4 Creating overlapping graphs

The following example creates two functions that are sampled into a two dimensional array. When this two dimensional array is plotted, each row is considered to be a separate graph, thus these graphs will be overlaid on top of each-other and distinguished by different colors.

```
// functions to sample:
in str funcA,funcB;
@funcA string "Func A";      // ex: sin(x)/x
@funcB string "Func B";      // ex: cos(x)

// min and max x values
in float x1,x2;              // ex: -10…10

// # of samples
float S = 256;

// create and sample
// output array
out float a[][2];
a[0] = sample(funcA,x1,x2,S);
a[1] = sample(funcB,x1,x2,S);

// setup graph interface
@a graph x1, x2, min(a), max(a) "Function Plot";
```

31 Charting

31.1 Pie Chart Output Interface

Pie chart considers sum of all array elements to be 100%. Subsequently each array element represents a slice of the pie chart. Please note that Pie charts do not function well in certain user interface configurations, in particular they will be unreadable in Series 60 and UIQ platforms unless they are maximized to fit the screen.

Interface Syntax:

```
@array pie [<legend>] ["<caption>"];

<legend> - opt. string array, pie chart legend (name of each slice).
<caption> - opt. string, pie chart caption.
```

Example:

```
out float a[] = [20 30 50];
@a pie;

out float a[] = [20 30 50];
@a pie ["First" "Second" "Third"] "Sample Pie Chart";
```

31.2 Bar Chart Output Interface

Each bar on the chart represents an array element. The height of the bar is determined by the specified maximum value in the bar interface parameters.

Interface Syntax:

```
@array bar [<max_value>] ["<caption>"];

<max_value> - opt. maximum bar value.
<caption>     - opt. bar chart caption.
```

Example:

```
out float a[] = [2 3 8 3 6 4 2 4 7];
@a bar 10 "Sample Bar Chart";

out float a[] = [2 3 8 3 6 4 2 4 7];
@a bar max(a) "Sample Bar Chart";
// uses max(a) to retrieve the largest array value
```

31.3 Histogram Output Interface

Histogram is a graphical representation of a frequency distribution. Each bar on the chart represents an integer value. The height of the bar is determined by the amount of times this value is encountered in the array.

Interface Syntax:

@array histogram [<max_value>] [“<caption>”];

<max_value> - opt. maximum bar value.
<caption> - opt. bar chart caption.

Example:

out float a[] = [2 2 8 3 6 2 8 4 7 1 2];
@a ***histogram*** *10 “Sample Histogram”;*

This example will create a bar chart indicating element iterations:
1:1 2:4 3:1 4:1 5:0 6:1 7:1 8:2 9:0 10:0

32 Date, Time and Duration

32.1 Date, Time and Duration Representation

F(x) provides functionality for date / time calculations. Date and Time values are stored in a 64 bit integer using internal Symbian SDK Time/Date format.

Internal Symbian time format represents a date and time as a number of microseconds since midnight, January 1st, 0 AD nominal Gregorian.

To store date/time values in the variable, you can use timestamp() function. To retrieve date/time elements from the variable you can use functions such as getyear(), gethour() etc.

32.2 Date and Time Interfaces

Date, Time and Duration values can be set via dynamic input variable interfaces.

Syntax:

```
@timestamp  date    ["<caption>"];      // offer date input
@timestamp  time    ["<caption>"];      // offer time input
@timestamp  datetime ["<caption>"];     // offer both date and time input
@timestamp  duration ["<caption>"];     // offer duration input
```

Example:

```
in int t1,t2,t3,t4;
@t1 date "Enter Date";
@t2 time "Enter Time";
@t3 datetime "Enter Date and Time";
@t4 duration "Enter Duration";
```

32.3 Duration Interface

Duration interface allows user to input the time duration by specifying hours, minutes and seconds (hours and minutes only on UIQ platform). The resulting value is represented in seconds.

Syntax:

```
@timestamp  duration ["<caption>"];     // offer duration input
```

Example:

```
in int d;
@d duration "Enter Duration";
```

33 Tables

Tables are regular CSV files that can be exported from or imported to spreadsheet capable applications like Microsoft Excel, Lotus 123 and Open Office.

33.1 File Format

CSV files are text files that contain rows and columns. One row of data is stored as a single line in a file and each column is separated with a comma. If a comma exists in the body of the column (the text itself), the column text will be enclosed in quotes. If a quote is present in the body of the text, it will be prefixed by another quote.

Example:

column one, ”column, two”, ”column ””three””

33.2 Column Names

In addition to the standard CSV file format convention F(x) allows creation of column names. Once column has been named, it is possible to access the column data by this name. Column names can be created by having the first row of the CSV file start with ‘#’ and contain comma separated names for each corresponding column.

Example:

```
#alpha, beta, gamma
40543,540452,3242
26571,4197,3852
```

Regardless of the column name row, row processing in F(x) is always zero based, so whether there is a column name row as first row or there isn’t, first row containing data is addressable as 0.

33.3 Data types

When F(x) reads a table, the data types of columns are determined dynamically. If F(x) encounters a number it will interpret it as a numerical value, if it encounters a string, it will be interpreted as string.

Table rows can frequently become multi-type arrays. Multi-type arrays are arrays that contain a mix of datatypes such as integer, floating point and string values. Multi-type arrays can only be used to store/retrieve information, majority of other functionality possible on arrays will produce errors.

Accessing of an element in a multi-type array is possible by dereferencing the array element using standard syntax “a[n]”; setting the array element in a multi-type

array may not be possible using standard dereferencing due to type mismatch. In this case you can use **set(<position>,<variant>)** function to set the value and enforce the type of the array element.

33.4 Table ID and Selection

Each opened table is represented by a unique id. Each opened table has a notion of the currently selected row. Once row is selected it is possible to access its data without having to specify the row number.

topen(<file name>) opens a table in the current directory.

Example:

int table_id = topen("records"); // .csv extension should not be specified
int table_id = topen("tables\records"); // specifies a table located in the subdirectory

Other essential functions:

tselect(<row number>) - selects the row as current.
tappend() - creates a new row and selects it as current.
trowget() - retrieves row data as an array
trowset() – sets row data from array into a table

The following example summates all values in a specified column from the CSV file A and stores the output into the CSV file B:

```
in bool button_calc;
@button_calc button "Calc Total";

int tid_sales = topen("sales_records");
int tid_totals = topen("sales_totals");

funds total = 0.0;     // our calculated total

// column number that contains values to be used
// in this case we use last column of the table
int col = tcolcount(tid_sales)-1;

for(int i = 0; i < trowcount(tid_sales); i++)
{
        funds row[] = trowget(tid_sales,i);
        total += row[col];

        // following syntax is possible but not used for clarity
        // total += trowget(tid_sales,i)[tcolcount(tid_sales)-1];
}

if(button_calc)
{
        // tappend() function creates new row and sets it as current
        // refer to tselect() function for more information
```

```
        tappend(tid_totals);
        trowset(tid_totals,total);
        notify("Total Calculated");
}
```

33.5 Table resources

Many table related functions use table resources to obtain table id and row information. The @..table interface allows selection of a table record from a pull-down list. The variable linked to the @..table interface automatically becomes a resource representing the current table and its selection.

Table resource is an array that is comprised out of the following information:

[id, row_number, total_row_count, selection_flag]

id – id number of an opened table
row_number – number of the currently selected row in the table row selection control
total_row_count – total number of rows in this table
selection_flag – indicates if selection has been changed

The selection_flags element is set to 1 when the applet execution is triggered by the record selection from the table row selection control.

34 Stacks

Stack is a temporary storage container that exists only during execution of the applet. Stack is a LIFO (Last In First Out) container, meaning that if you place items A B C on the stack when you retrieve them the order of the items will be C B A.

Stack functions utilize handles to differentiate stacks. Stack handle can be any numerical value as stacks are created dynamically. This means that if you use value "1" a stack with such handle will be created automatically for you.

Example:

```
for(int i = 0; i < 100; i++)
{
    if(i & 0x1) push(123,i); // place all odd numbers on the stack
}

i = 0;

while(stackcount(123))
        i += pop(123);        // add all odd numbers from stack

i;      // display result
```

35 Dynamic User Interface Creation

F(x) offers functions for dynamic creation of user interface. Typically user interface is built by analyzing applet variables and their interfaces, dynamic UI function allow conditional creation of variables. This functionality is especially helpful for creation of interactive forms (applets used to gather and then store/submit data).

35.1 Variable Access

Following example uses user interface functions to analyze the applet environment and build a human readable string that can then be transmitted to a 3rd party via SMS or E-Mail:

```
in str client;                  // name of the client
@client string "Client Name";

in bool q1,q2,q3,q4,q5,q6;      // manually created variables

// a title can be assigned to the variable at the interface creation time
// @<name> <interface> “<title>”

@q1 check "Diabetes";           // variable titles
@q2 check "Heart Disease";
@q3 check "Mental Illness";
@q4 check "High BP";
@q5 check "Cancer";
@q6 check "Epilepsy/Seisure";

// message string to be generated by this applet
out str msg = "Family History for: "+client+cr();
// cr() function adds a carriage return (new line)

// main loop that iterates through sequential variables starting with "q"
// vpresent() function tests for existence of the variable
// itoa() function is used to convert integer i to string
for(int i = 1; vexists("q"+i); i++)
{
        // vtitle() function retrieves the title of the variable by name
        // vvalue() retrieves a value of the variable by name

        msg += vtitle("q"+i)+": " +(vvalue("q"+i) ? "YES" : "NO")+"\n";
}

// create "SEND" output interface for the string msg.
// SEND output interface will display a button, upon hitting which the
// contents of the string will be transmitted via E-Mail, SMS or another
// method selected by the end-user.

@msg send "Send History Info";
```

35.2 Variable Creation

The following example demonstrates dynamic creation of the user interface based on table data.

```
// reset all previous dynamically created variables
uireset();

// tselect interface lists all tables residing in the same folder as the
// current applet and allows user to select one of them. In this example
// the selected table will be used as a source of variable titles.
in str tbl;
@tbl tselect;

// if no table is selected, display an error and abort
if(!strlen(tbl)) error("PLEASE SELECT TABLE");

// for the purpose of the example we create a progress bar that
// displays how many percent of variables have been marked checked.
out float total;
@total progress;

int t = topen(tbl);     // open the user selected table

str a[];
a = tcolget(t,0);       // retrieve first column of the table as an array

out str m;              // message to be generated for output/sending

for(int i = 0; i < count(a); i++)
{
        // vcreatein(), vcreateout() and vcreate() allow creation of
        // input, output and local variables of a given type.

        // create integer variables with name and title of a[i]
        vcreatein("int",a[i],a[i]);

        // assign created variable a checkbox interface
        viface(a[i],"check");
}

// update user interface to reflect created variables
uiupdate();

// scan created variables for values and build a text message
for( i = 0; i < count(a); i++)
{
        total += vvalue(a[i]) ? 1:0; // count each variable set to TRUE
        m += vtitle(a[i]) + ": " + (vvalue(a[i]) ? "YES" : "NO") +" \n";
}
```

```
// set the total value as percentage of variable count
total = total / count(a) * 100;

// assign variable m a SEND interface so that
// it can be dispatched to the messaging system.
@m send "SEND RESULT";
```

35.3 F(x) User Interface Paradigm

A developer reviewing the above example will immediately ask the following question: “How can we create a variable for user input and immediately retrieve the value entered by the user?”.

As you edit applets on device or when you exit and open the same applet you will notice that input values are preserved by F(x). Editing of applets frequently produces syntax errors, when these errors are corrected F(x) restores original user input.

The user input caching mechanism is one of the components that allows dynamic user interface construction to function correctly. Once you create an input variable N and the user inputs a value in it, the value is cached. When the applet rebuilds the user interface and reintroduces the variable N, F(x) will retrieve its value from cache upon creation.

35.4 Variable Content Modification

It is possible to modify the contents of the variable input interface, provided that this interface is of EDIT or STRING type.

Function setiv(<variable name>,<variable value>) allows user to set the input edit field to specific text. Please note that this will occur for every applet invocation, thus this type of functionality has to be conditional:

```
in int multiplier;      // input variable

in bool reset_btn;     // reset button
@reset_btn button “Reset”;

if(reset_btn)
{
        // if button pressed, reset default value to 0.5
        setiv(“multiplier”, “0.5”);
}
```

36 Executing Applets as Functions

Currently F(x) does not offer ability to create in-applet functions, however creation and invocation of user functions can be achieved in two ways.

First way is the already mentioned method of value sampling, where sample() function is invoked with the body of a user function specified as a string parameter.

Second way is to invoke another applet as a user function using exec(). Unlike sample(), the applet executed via exec() does not have access to the current applet workspace, thus it can not use variables present in the invoking applet.

exec(<string applet name>, <parameter array>) function allows execution of an external applet and returns the result of the applet. (External applet execution is very fast because the invoked applet if precompiled into F(x) byte code.)

Array of parameters specified in the second argument to exec() is passed on to the invoked applets as an array variable "argv". When creating an applet to be used as a function, you can declare the argv[] variable to simulate valid parameter input. Once the applet invoked, the contents of the argv[] array will be replaced with those supplied to the exec() function. If argv[] variable does not exist, it will be created by the compiler.

Example:

```
// ~~~
// child applet.f(x)

// this applet will be invoked as a function.
// implementing f(x) = a*x^2+b*x+c

float argv[4]; // x a b c

// this is the last operation on the stack,
// thus it is considered the final applet result.

argv[1] * sqr(argv[0]) + argv[2] * argv[0] + argv[3];

-

// ~~~
// parent applet.f(x)

out float a[32];
for(int i = 0; i < count(a); i++)
  a[i] = exec("child applet",[i 1 1 1]);
```

Please note that recursive execution of applets using exec() function is not possible and a recursion error will be reported if user attempts to do so.

37 Continuous Execution

It is possible to make an applet run continuously. This can be achieved using respawn() function. This function will schedule the applet to be executed again once it is finished running.

This ability can be useful when using F(x) to accumulate real-time user interaction or represent time relative to current.

Example:

```
int t = hometime();
respawn();
t;
```

The above example will continuously update the timestamp value as the result of the applet until the user switches to another application or exits the applet.

38 F(x) Applet Library

F(x) comes with a package that contains various applets and formulas for use with F(x). While this package is included with F(x), it is also available for installation separately. This applet package is a standalone SIS (Symbian Installation System) file, and can be upgraded separately from F(x).

IMPORTANT – When upgrading F(x) Applet Library, Symbian Installer will remove any previously installed files. Consequently if you have modified any applets that are a part of the F(x) Applet Library, these changes will be lost. You must rename the files you have modified or move them into a different folder in order for the Symbian installer to ignore them. The installer will not touch any of the files that were not a part of the previous installation.

39 Support

http://developer.symbianfx.com web site contains additional information related to the F(x) software, its updates, development information and support forums on which you may post any questions, your applets or ideas for further software development.

If you have purchased F(x) and have a specific technical question, you may e-mail it to support@symbianfx.com.

You may also inform us of any problems you encounter by submitting applets with comments via applet submission function (available via F(x)/Submit menu). Please make sure to include your e-mail address in the applet source code as a comment so that we may get back to you.

40 Applet Submission

Applets created by you can be submitted to the symbianfx.com site to be made available to other F(x) users. Once submitted, the applet will be reviewed and posted in the appropriate site category. The applet may be modified in order to clarify or enhance its functionality.

Applet submission is anonymous, no information pertaining to the user or the device will be submitted during the process, unless the user chooses to include his/her credentials in the applet itself in the form of the comment. If done so, this comment will be preserved in the applet as it is placed online.

In order to submit the applet, you must have a GPRS or an alternate method of the internet connection directly from the Symbian device running F(x). Applet submission can be done via “F(x)/Submit” menu. Alternatively, you may submit applets by e-mailing them to submit@symbianfx.com or posting them on the discussion forum available at http://developer.symbianfx.com/forum/ .

When submitting an applet, please use comments to communicate any additional information, such as the description and application of the applet.

41 Software License Agreement

OWNERSHIP OF SOFTWARE

You acknowledge and agree that this software solution (named “F(x)”) and associated documentation (collectively, the "Software"), except applets included with or designed for the software, are owned exclusively by Loginid Enr. You agree that the price paid by you for the Software is a license fee granting you only the rights set forth in this License Agreement.

LICENSE

Loginid Enr., grants to you, and you accept, a limited, non-exclusive and revocable license to use the Software, in machine-readable, object code form only. You agree to use the Software only as authorized in this License Agreement. This License Agreement does not convey to you any ownership rights or any other interest in the Software.

SCOPE OF LICENSE

This Software is licensed to be installed and used on only one computer. A valid license must be purchased for each computer on which the Software is installed.

You may not copy or make any changes or modifications to the Software, and you may not translate, decompile, disassemble, or otherwise reverse engineer the Software. You may not lend, rent, lease or sublicense the Software or any copy to others for any purpose. You agree to use all reasonable efforts to protect the Software from unauthorized use, modification, reproduction, distribution or publication. You are not permitted to make any uses or copies of the Software that are not specifically authorized by the terms of this License Agreement, and Loginid Enr., reserves all rights that are not expressly granted to you. Your adherence to this License Agreement will allow Loginid Enr., to continue developing innovative and useful products and providing a high level of customer service and support.

TERM

This license will become effective on the date you acquire the Software and will remain in force until terminated. You may terminate the license at any time by removing the Software from your computer and destroying the original Software and all copies. This license will automatically terminate if you breach any of the terms or conditions set out in this License Agreement. You agree to remove the Software from your computer, and either to destroy the original Software and all copies of the Software or to return the Software to Loginid Enr., upon termination of this license for any reason.

LIMITATIONS OF LIABILITY AND REMEDIES

In no event shall Loginid Enr., or its licensors be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, punitive or other damages, even if Loginid Enr., or its licensors are advised, in advance, of the possibility of such damages. In no event shall the liability of Loginid Enr., or its licensors exceed the purchase price paid for the Software. In no event shall Loginid Enr., or its licensors be liable for the content of applets or algorithms used in them, whether included with or created for the software.

OTHER TERMS AND CONDITIONS

You agree to inform anyone who you may record that their Internet and Computer Activity is subject to being recorded and archived. You agree to install this software ONLY on a computer that you own or on a computer which you have been given explicit permission to install. You agree to NOT install this software on any computer you do not own or on any computer you have not been given explicit permission to install.

Appendix A – Simple Examples

F(x) as a tape / financial calculator

F(x) statements are separated by the semicolon character. If the semicolon character is not present, entered expression is considered to be a single statement.

Examples:

```
435.57+
34.90+
764.34+
634.94
```

Above expression will add the numbers together and display the result.

F(x) as a tape calculator with tax

To calculate taxes for the above amount this applet can be modified as follows:

```
out funds Sub_Total =

435.57+
34.90+
764.34+
634.94;

in real Tax_Rate;
out funds Tax = Sub_Total * Tax_Rate;
out funds Total = Sub_Total + Tax;
```

Above applet allows to enter values that comprise Sub_Total amount into the applet itself. Since the values are entered in the applet editor itself, they can be entered one value per line.

Alternative method that can be used to enter the values for the Sub_Total amount, is by use of dynamic input variables:

```
in funds Sub_Total;
in real Tax_Rate;
out funds Taxl = Sub_Total * Tax_Rate;
out funds Total = Sub_Total + Tax;
```

In the dynamic variable input you can specify the values as follows:

```
Sub_Total $: 435.57+34.90+764.34+634.94
```

The following example is provided to show how to calculate taxes in the country with dual taxation such as Canada:

```
in funds Sub_Total;
real Federal_Rate = 7.0;     // Canadian Federal Tax Rate
real Provential_Rate = 7.5; // Tax Rate for the province of Quebec
out funds Provential_Tax = Sub_Total * Provential_Rate;
out funds Federal_Tax = (Sub_Total + Provential_Tax) * Federal_Rate;
out funds Total = Sub_Total + Provential_Tax + Federal_Tax;
```

Base Conversion

The following applet demonstrates how F(x) can be used for conversion between numerical representations of values:

```
in int Value;
out bin Binary = Value;
out oct Octadecimal = Value;
out dec Decimal = Value;
out hex Hexadecimal = Value;
```

By declaring 4 variables as dynamic output variables of types bin,oct,dec,hex the output for each variable will have corresponding numerical representation.

When entering the value of the dynamic input variable “Value” you can use different numerical representations (please see Section 16 - Numerical Representations for more information).

Example:

```
Value: 0b1001+0xABCD+0c4545
```

Appendix B – Unit conversion

Unit Abbreviations used in F(x):

AREA

Ac - acres, **Ares** - ares, **CIn** - circular inches, **Ha** - hectares, **Hides** - hides (with variations), **Rod** - roods, **CmSq** - square centimetres, **FtSq** - square feet (UK and US), **FtSqSrv** - square feet (US survey), **InSq** - square inches, **KmSq** - square kilometres, **MSq** - square metres, **MiSq** - square miles, **MmSq** - square millimetres, **RdSq** - square rods (or poles), **YdSq** - square yards, **TimberSq** - squares (of timber), **Twp** - townships

DENSITY

GrnGalUK - grains/gallon(UK), **GrnGalUS** - grains/gallon(US), **GCmCu** - grams/cubic centimetre, **GL** - grams/litre, **GMl** - grams/millilitre, **KgMCu** - kilograms/cubic metre, **KgL** - kilograms/litre, **MegaGrPerCuM** - megagrams/cubic metre, **MgL** - milligrams/litre, **MgMl** - milligrams/millilitre, **OzInCu** - ounces/cubic inch, **OzGalUK** - ounces/gallon(UK), **OzGalUS** - ounces/gallon(US), **LbFtCu** - pounds/cubic foot, **LbInCu** - pounds/cubic inch, **LbGalUK** - pounds/gallon(UK), **LbGalUS** - pounds/gallon(US), **TMCu** - tonnes/cubic metre, **TUKYdCu** - tons(UK)/cubic yard, **TUSYdCu** - tons(US)/cubic yard

ENERGY

Btu - British Thermal Units (International Table), **BtuMean** - British Thermal Units (mean), **BtuTC** - British Thermal Units (thermochemical), **CalFood** - Calorie (food) (approx.), **Cal15C** - Calories (15C), **Cal20C** - Calories (20C), **Cal** - Calories (International Table), **CalMean** - Calories (mean), **CalTC** - Calories (thermochemical), **C** - centigrade heat units, **Erg** - ergs, **FtPl** - foot poundals, **FtLb** - foot pounds-force, **GJ** - gigajoules [GJ], **HpHr** - horsepower hours (approx.), **J** - joules [J], **KCal** - kilocalories (IT) (International Table), **KCalTH** - kilocalories (th), **kJ** - kilojoules [kJ], **kWHr** - kilowatt hours [kWh], **MJ** - megajoules [MJ], **Thm** - therms (approx.), **WHr** - watt hours [Wh], **WSec** - watt seconds [Ws]

FORCE

Dyn - dynes, **KgF** - kilograms force, **kN** - kilonewtons [kN], **Kip** - kips, **MN** - meganewtons [MN], **N** - newtons [N], **Pl** - poundals, **LbF** - pounds force, **Sn** - sthenes (=kN), **TF** - tonnes force, **TFUK** - tons(UK) force, **TFUS** - tons(US) force

LENGTH

A - angstroms, **AU** - astronomical units, **Brc** - barleycorns, **Cm** - centimetres, **Ch** - chains (surveyors'), **Ems** - ems (pica), **Fth** - fathoms, **Ft** - feet (UK and US), **FtSrv** - feet (US survey), **Fur** - furlongs, **Hands** - hands, **In** - inches, **Km** - kilometres, **Lg** - leagues (4000 to 5000), **Ly** - light years, **Link** - links (surveyors'), **M** - metres [m], **Mc** - microns (=micrometres), **Nmi** - miles (nautical), **Mi** - miles (UK and US), **Pc** - parsecs, **Perch** - perch (=rods or poles), **PiC** - picas (computer), **PiPr** - picas (printers'), **PtC** - points (computer), **PtPr** - points (printers'), **Yd** - yards

LINE DENSITY

Denier - denier, **Drex** - drex, **GCm** - grams/centimetre, **GKm** - grams/kilometre (tex), **GM** - grams/metre, **GMm** - grams/millimetre, **KgKm** - kilograms/kilometre, **KgM** - kilograms/metre, **MgCm** - milligrams/centimetre, **MgMm** - milligrams/millimetre, **OzFt** - ounces/foot, **OzIn** - ounces/inch, **LbFt** - pounds/foot, **LbIn** - pounds/inch, **LbMi** - pounds/mile, **LbYd** - pounds/yard, **Tex** - tex, **TKm** - tonnes/kilometre, **TUKMi** - tons(UK)/mile, **TUSMi** - tons(US)/mile

POWER

BtuHr - Btu/hour, **BtuMin** - Btu/minute, **BtuSec** - Btu/second, **CalHr** - calories/hour, **CalMin** - calories/minute, **CalSec** - calories/second, **FtLbMin** - ft lb-force/minute, **FtLbSec** - ft lb-force/second, **GW** - gigawatts [GW], **HpE** - horsepower (electric), **HpM** - horsepower (metric), **JHr** - joules/hour, **Jmin** - joules/minute, **Jsec** - joules/second, **KgMHr** - kg-force metres/hour, **KgMMin** - kg-force metres/minute, **KCalHr** - kilocalories/hour, **KCalMin** - kilocalories/minute, **kW** - kilowatts [kW], **MW** - megawatts [MW], **W** - watts [W]

PRESSURE OR STRESS

Atm - atmospheres, **B** - bars, **CmHg** - centimetres of mercury, **CmH2O** - centimetres of water, **FtH2O** - feet of water, **hPa** - hectopascals [hPa], **InHg** - inches of mercury, **InH2O** - inches of water, **KgCmSq** - kg-force/sq.centimetre, **KgMSq** - kg-force/sq.metre, **KNMSq** - kilonewton/sq.metre, **kPa** - kilopascal [kPa], **KipInSq** - kips/sq.inch, **MNMSq** - meganewtons/sq.metre, **MH2O** - metres of water, **Mb** - millibars, **MmHg** - millimetres of mercury, **MmH2O** - millimetres of water, **NCmSq** - newtons/sq.centimetre, **NMSq** - newtons/sq.metre, **NMmSq** - newtons/sq.millimetre, **Pa** - pascals [Pa], **PlFtSq** - poundals/sq.foot, **LbFtSq** - pounds-force/sq.foot, **LbInSq** - pounds-force/sq.inch, **TCmSq** - tonnes-force/sq.cm, **TMSq** - tonnes-force/sq.metre, **TUKFtSq** - tons(UK)-force/sq.foot, **TUKInSq** - tons(UK)-force/sq.inch, **TUSFtSq** - tons(US)-force/sq.foot, **TUSInSq** - tons(US)-force/sq.inch

SPEED

CmMin - centimetres/minute, **CmSec** - centimetres/second, **FtHr** - feet/hour, **FtMin** - feet/minute, **FtSec** - feet/second, **InMin** - inches/minute, **InSec** - inches/second, **KmHr** - kilometres/hour, **KmSec** - kilometres/second, **Kt** - knots, **Ma** - Mach number, **MHr** - metres/hour, **MMin** - metres/minute, **MSec** - metres/second [m/s], **MiHr** - miles/hour, **MiMin** - miles/minute, **MiSec** - miles/second, **YdHr** - yards/hour, **YdMin** - yards/minute, **YdSec** - yards/second

SPREAD RATE (MASS)

GCmSqSr - grams/sq.centimetre, **GMSqSr** - grams/sq.metre, **InRainMSr** - inches of rainfall, **KgHaSr** - kilograms/hectare, **KgCmSqSr** - kilograms/sq.centimetre, **KgMSqSr** - kilograms/sq.metre, **MgMSqSr** - milligrams/sq.metre, **MmRainMSr** - millimetres of rainfall, **OzFtSqSr** - ounces/sq.foot, **OzInSqSr** - ounces/sq.inch, **OzYdSqSr** - ounces/sq.yard, **LbAcSr** - pounds/acre, **LbFtSqSr** - pounds/sq.foot, **LbInSqSr** - pounds/sq.inch, **LbYdSqSr** - pounds/sq.yard, **THaSr** - tonnes/hectare, **TUKAcSr** - tons(UK)/acre, **TUSAcSr** - tons(US)/acre

SPREAD RATE (VOLUME)

FtCuAc - cubic feet/acre, **InCuYdSq** - cubic inches/sq.yard, **MCuHa** - cubic metres/hectare, **MCuKmSq** - cubic metres/sq.km, **MCuMSq** - cubic metres/sq.metre, **YdCuMiSq** - cubic yards/sq.mile, **OzFlYdSq** - fl. ounces(UK)/sq.yard (UK), **GalUKAc** - gallons(UK)/acre, **GalUKHa** - gallons(UK)/hectare, **GalUSAc** - gallons(US)/acre, **GalUSHa** - gallons(US)/hectare, **InRainV** - inches of rainfall, **Lha** - litres/hectare, **LMSq** - litres/square metre, **MlMSq** - millilitres/sq.metre, **MmRainV** - millimetres of rainfall

TORQUE

DynCm - dyne centimetres, **GFCm** - gram-force centimetres, **KgFCm** - kg-force centimetres, **KgFM** - kg-force metres, **NCm** - newton centimetres, **Nm** - newton metres [Nm], **OzFIn** - ounce-force inches, **LbFFt** - pound-force feet, **LbFIn** - pound-force inches, **PlFt** - poundal feet, **TFUKFt** - ton(UK)-force feet, **TFUSFt** - ton(US)-force feet, **TM** - tonne-force metres

VOLUME AND CAPACITY

Brl - barrels (oil), **BuUK** - bushels (UK), **BuUS** - bushels (US), **Cl** - centilitres, **CC** - cubic centimetres, **DamCu** - cubic decametres, **DmCu** - cubic decimetres, **FtCu** - cubic feet, **InCu** - cubic inches, **Mcu** - cubic metres, **MmCu** - cubic millimetres, **YdCu** - cubic yards, **DL** - decilitres, **OzUS** - fluid ounces (UK), **OzUK** - fluid ounces (US), **GalUK** - gallons (UK), **GalDUS** - gallons, dry (US), **GalLUS** - gallons, liquid (US), **LOld** - litres (1901 - 1964), **L** - litres [l or L], **Ml** - millilitres, **PtUK** - pints (UK), **PtDUS** - pints, dry (US), **PtUS** - pints, liquid (US), **QtUK** - quarts (UK), **QtDUS** - quarts, dry (US), **QtLUS** - quarts, liquid (US)

MASS OR WEIGHT

Ct - carats, metric, **Grn** - grains, **G** - grams, **CwtL** - hundredweights, long, **CwtS** - hundredweights, short, **Kg** - kilograms, **OzAv** - ounces, avoirdupois, **OzT** - ounces, troy, **Lb** - pounds, **Slugs** - slugs (or g-pounds), **Stones** - stones, **T** - tonnes, **TUK** - tons (UK or long), **TUS** - tons (US or short)

TIME

Day - day, **Hr** - hour, **Mcs** - microsecond, **Ms** - milliseconds, **Min** - minute, **Sec** - second, **Wk** - week, **Y** - year

International System of Units (SI)

All systems of weights and measures, metric and non-metric, are linked through a network of international agreements supporting the **International System of Units**. The International System is called the **SI**, using the first two initials of its French name *Système International d'Unités*. The key agreement is the Treaty of the Meter (*Convention du Mètre*), signed in Paris on May 20, 1875. 48 nations have now signed this treaty, including all the major industrialized countries. The United States is a charter member of this metric club, having signed the original document back in 1875.

F(x) Applet Library package that is included in the installation comes with a number of SI unit conversion constants. These constants cover measurements such as *Length, Area, Volume/Capacity, Mass/Weight, Density, Energy, Force, Power, Pressure/Stress, Speed, Spread Rates (Mass and Volume), Torque and Fuel Consumption*.

To convert a value to an SI unit of measurement you must multiply this value by the SI conversion constant. For example, to convert Feet to SI unit of length, you must multiply Feet by Ft2SI constant.

Conversion Between Units

To convert one unit to another, using SI constants, one can use automated conversion constants that are calculated by F(x) during the applet execution.

For example: to convert Feet to Meters you should use Ft2M constant. This constant is automatically calculated at the compile time by establishing a relationship between two constants as follows: Ft2M = Ft2SI/M2SI. Any units can be mixed using XXX2XXX format as long as they belong to the same category.

Appendix D – Technical Information

Floating Point Resolution:	64 bit double precision floating point, approx. 2.225074 × 10^{-308} to 1.797693 × 10^{+308}, with accuracy of 15 decimal places (IEEE754).
Integer Resolution:	64 bit, ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
Operating System:	Symbian OS running following user interface platforms: • UIQ 2.1 • Series60 • Series80 • Series90
Startup memory requirements:	Approx. 500 KB.
Storage requirements:	Approx. 1MB.