Mint Project User Guide:

This User Guide is intended to help developers understand the purpose, layout, and function of Mint Project 1.0. This guide is intended for new programmers, as well as programmers who are familiar with current programming languages (specifically Objective-C, and JavaScript). Like Mint Project, this User Guide is a work in progress, please check <http://www.mintsoftware.com> for updates.

Contents:

1. What is Mint Project:

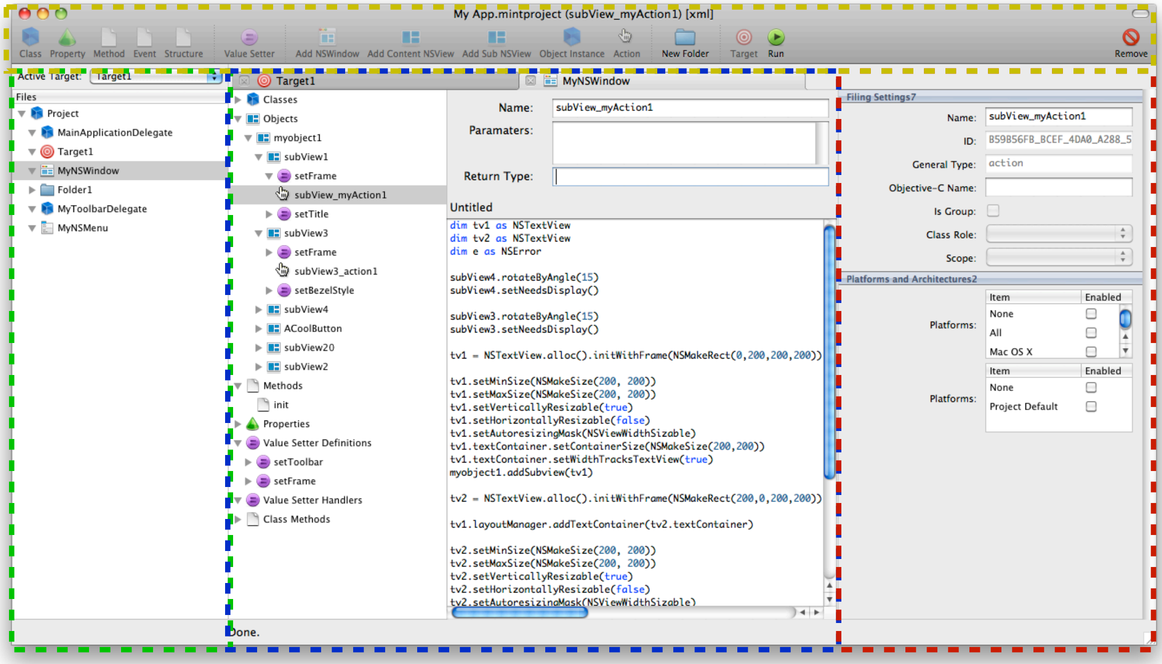
Mint Project is designed to aid in Application Development, currently for the Mac OS X platform, tho other platforms are expected to be supported, as there is a lot of interest in iOS (previously iPhone OS) and HTML5/JavaScript. At its core, Mint Project is a ‘project organizer’ and IDE (Integrated Development Environment).

a. Application Creation Tool

Mint Project is designed to truly cater to the needs of developers. Currently, developers of Mac OS X, iOS, et cetera create programs in Objective-C language and Xcode which require the usage of Header (.h) and Implementation (.m) files, usually with each file holding the code for an individual class or group of related classes. For small programs, this is not a problem; however, for larger projects where there are many hundreds of classes and functions and methods, this can become a headache. Mint Project solves that problem by allowing you to visually organize classes, methods, class methods, and properties instead of having to look at pages and pages of code or flipping between Header and Implementation files (See The Interface).

Mint Project does not use the Objective-C language, instead it takes advantage of BASIC (Mint Basic); see Section 4 for more information on Mint Basic.

2. The Interface:



Name: subView_myAction1
Paramaters:
Return Type:

Untitled

```
dim tv1 as NSTextView
dim tv2 as NSTextView
dim e as NSError

subView4.rotateByAngle(15)
subView4.setNeedsDisplay()

subView3.rotateByAngle(15)
subView3.setNeedsDisplay()

tv1 = NSTextView.alloc().initWithFrame(NSMakeRect(0,200,200,200))

tv1.setMinSize(NSMakeSize(200, 200))
tv1.setMaxSize(NSMakeSize(200, 200))
tv1.setVerticallyResizable(true)
tv1.setHorizontallyResizable(false)
tv1.setAutoresizingMask(NSViewWidthSizable)
tv1.textContainer.setContainerSize(NSMakeSize(200,200))
tv1.textContainer.setWidthTracksTextView(true)
myobject1.addSubview(tv1)

tv2 = NSTextView.alloc().initWithFrame(NSMakeRect(200,0,200,200))

tv1.layoutManager.addTextContainer(tv2.textContainer)

tv2.setMinSize(NSMakeSize(200, 200))
tv2.setMaxSize(NSMakeSize(200, 200))
tv2.setVerticallyResizable(true)
tv2.setHorizontallyResizable(false)
tv2.setAutoresizingMask(NSViewWidthSizable)
```

Filing Settings7

Name: subView_myAction1
ID: B59B56FB_BCEF_4DA0_A288_5
General Type: action
Objective-C Name:
Is Group: ☐
Class Role:
Scope:

Platforms and Architectures2

Platforms:

Item	Enabled
None	☐
All	☐
Mac OS X	☐

Platforms:

Item	Enabled
None	☐
Project Default	☐

The interface is composed by regions of panes that adjust themselves to the currently selected project item; however it can be generalized into four different regions.

1. The Toolbar (yellow) which contains all the buttons necessary to the project.
2. The Project Browser (green), which contains all the classes, folders, and targets.
3. Tabbed Area (blue) which allows you to have several classes, targets, or build result panels open.
4. Member Inspector (red) which adjusts its self as you select items.

The most versatile region is the *Tabbed Area* which shows the most important pane, a class editor which is where the real work gets done in Mint Project. It allows you to organize a class and its members. In this panel you are allowed to add methods (or class methods), properties, 'events' and 'objects,' 'value setters,' 'object actions.'

3. Targets, and Classes, and Members, Oh My:

This section describes the two main project members: Targets and Classes (and class members).

a. Targets:

Targets are items which can be added to a project and allow you to setup the specifications for a build-product. Currently, a target holds the specifications for Platform, Architecture, Frameworks, Compiler, Active Classes, and Resources.

Two supported platforms are expected to make it into the final version of Mint Project: Mac OS X, iOS (Formally iPhone OS), and HTML5 being added later. For each of these platform types you can select which architecture (processor) can or should be used, which classes should be included during compile, and which resources should be added to the final product (Application or Framework bundle). Each of these features work slightly different for each platform (especially for HTML5).

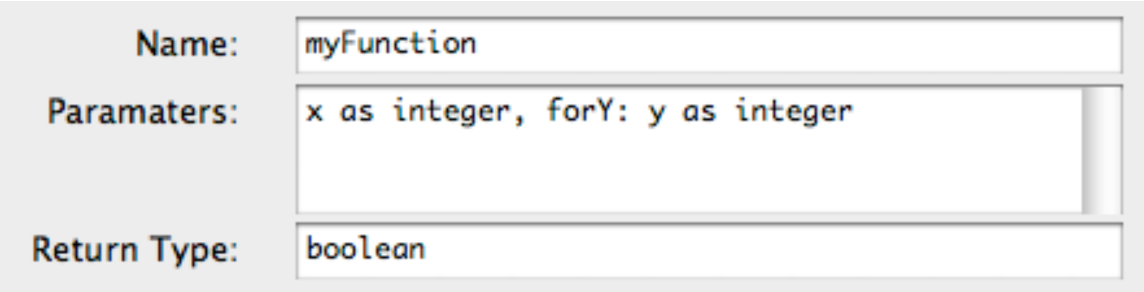
- **Notes about Mac OS X and iOS**: On these two platforms, Mint Project creates executables which use the *Objective-C runtime*. A runtime is not a language, however it does implement concepts which parallel the Objective-C language. These paradigms are discussed in the *Section 4: Mint Language* and in the next subsection *(b)*. It is important to understand that altho Mint Basic is not Objective-C it must adopt particular features in order to sensibly interface with the *Obj-C Runtime*.

b. Classes and Their Members (*Within the context of the Objective-C runtime*)

This section details the specification for Classes and how they function on particular Platforms. This information is useful for those who program in Objective-C and wish to understand how Mint Project has compensated for certain design paradigms in Objective-C and the *Objective-C (ObjC) Runtime*.

The Cocoa environment on Mac OS X or iOS uses the *Objective-C Runtime*, thus, programs generated with Mint Project must work well with this runtime. When classes are generated for the *Objective-C Runtime*, they inherit all the abilities, rights, and responsibilities of an Objective-C class (just as it would be if it was written directly in Objective-C).

- **Properties** are instance variables (Ivar) that are coupled with a property (in the sense of Objective-C's `@property`, `@synthesize` directives).
- **Methods** work just as they do in Objective-C, selectors and all. When adding a method, you are given the ability to name the parameters as in the following example:



Name: myFunction

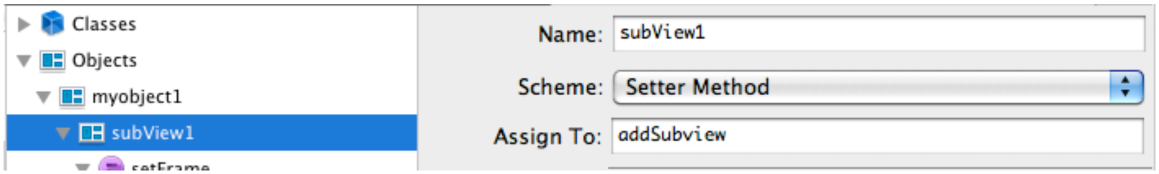
Paramaters: x as integer, forY: y as integer

Return Type: boolean

This method's selector will be *myFunction:forY:*

There are several member types in Mint Project which get special considerations. Four of them are detailed here.

- **Objects** are instances of a class which are assigned to an Instance Variable (ivar) and also have a property to access them. They work just as properties work except they are instantiated at runtime and can be sent to a method (owned by the encompassing class that take one parameter). A good example of this is a Object whose super class is a NSView who needs to be set to "addSubview". (In Mint Project, any class who is sent to the method addSubview is visible in the Layout Editor). Objects generate the following code (this object has been set to be a NSView in another panel). In earlier betas of Mint Project Objects were completely owned by the class on which they are implemented; however, as of beta 6, objects are owned by their containing object, and have to be accesssed thru their containing object (e.g. MyWindow.myContentView.NSButton1.press()).



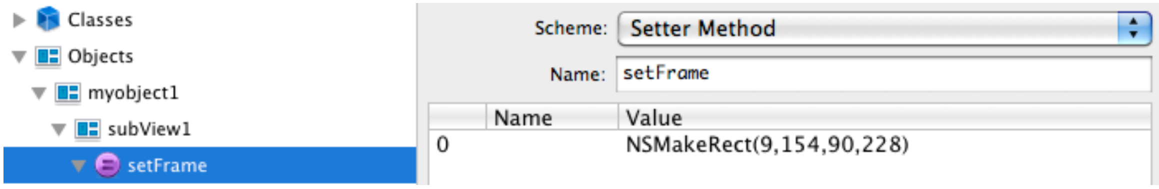
Name: subView1

Scheme: Setter Method

Assign To: addSubview

```
...//the following code exists in the declaration of the Class, and not in a function.
subView1 as NSView // (instance variable)
...//this code appears in a method that is auto generated by the Mint IDE, a method that is not visible to the user.
subView1 = NSView.alloc().init()
self.addSubview(myNSView)
```

- **Value Setters** are special members that can be added to a class, or object. Their function is to call a method on that object after its creation. A Value Setter is given a method, and group of parameters to send to that method. They create the equivalent of the following code (in blue)



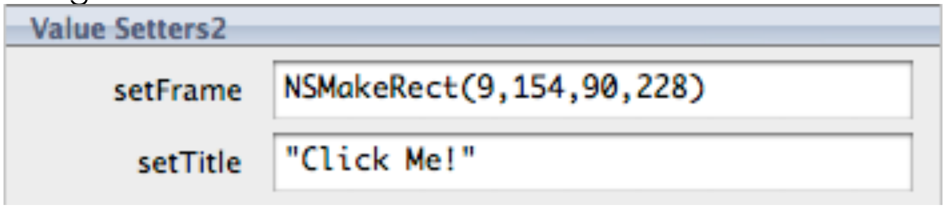
Scheme: Setter Method

Name: setFrame

	Name	Value
0		NSMakeRect(9,154,90,228)

```
subView1 = new NSView
subView1.setFrame(NSMakeRect(10,10,22,60)
self.addSubview(subView1)
```

Value Setters who have only one parameter also appear in the Member Inspector on the right:



setFrame NSMakeRect(9,154,90,228)

setTitle "Click Me!"

- **Object Actions** are basically methods which are owned by the class but are assigned to the object under which it is made. Such methods can only be used with certain classes but currently Mint Project will not block you from adding an action message to a object. Classes with which action messages can be used define two methods *setAction:* and *setTarget:*. In this case the target is the class, and the action is the Object Action.

- **Events (events are currently not working in the betas)** are a little more complicated. Unlike Methods that can be overridden completely by a subclass, Events are intended to be able to pass messaged starting from the lowest super-class up the sub-class chain. Essentially they are methods that are completely controlled by Mint Project to simulate this effect, the user should not need to understand how it is implemented.

```
raiseEvent mouseDown(x,y)
//or
myObject.raiseEvent mouseDown(x,y)
```

4. Mint Basic Language

Mint Basic is an implementation of BASIC. Users who have programmed in products such as Real Studio (formally REALbasic) or Visual Basic, with some exceptions, should be right at home in Mint Project.

a. Intrinsic Datatypes:

Mint Basic supports most of the datatypes that Objective-C or C support such as integer (int), boolean, double, IMP, id, SEL, et cetera. Mint Project attempts to tightly control these primitive types and does not allow the creation of new primitive types (tho, the creation of structure might be supported later).

> **For Advanced Users:** The design decision to not allow creation of new primitive types in Mint Project is due to the fact that as it is, the Objective-C framework is full of redundant redefinitions of types, so much so that it becomes crazy to deal with. The creation of structures will most likely be supported in later version.

b. Declaring Variables

In Mint Basic, variables are declared using the following syntax

```
// Mint Basic
dim i as integer
dim myNSView as NSView
```

c. Setting Variables or Properties, and calling Methods

```
i = 10
myNSView = NSView.alloc().initWithFrame(NSMakeRect(0,0,100,100))
```

d. If…Then Statements

```
if i=10 or i=0 then
//some code
elseif i=3 then
//some code
end if
```

e. For…While…Using…Next Loop

```
for i=10 while i<100 using i=i+1
//some code
next
```

f. While…Wend

```
while i<10
//some code
wend
```

There are, however, some differences between Mint Basic and other implementations of BASIC. Such differences were necessary to make Mint Basic function in an Objective-C environment.

One such difference stems from the fact that the *Objective-C runtime* uses 'selectors' to send messages which invoke a particular implementation for a method. Selectors provide the feature to name parameters, thus Mint Basic needs to allow naming parameters in order to interface properly with the Objective-C runtime. Examine the following code:

```
// Mint Basic
myNSView.addToolTipRect(aRect, owner: anObject, userData: userData)
```

```
// Objective-C
[myNSView addToolTipRect:aRect owner: anObject userData: userData]
```

g. Other Keywords

Mint Basic has many keywords including new, raiseEvent, return, self, me.