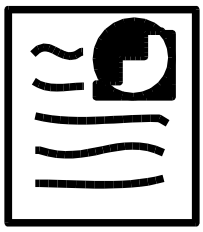


StepTalk

The Scripting Framework

Stefan Urbanek

Note: This is a draft of StepTalk Document. At this moment it is a compilation of various short documents that need to be edited.

Contents

1 Introduction

StepTalk is the official GNUstep scripting framework. The goal of StepTalk is to provide an environment for gluing applications together and to provide a framework for communication between users and objects in the applications. The framework allows one, besides other things, to:

- simply and quick extensid applications or tools
- batch-process objects in applications or tools
- fast prototype applications or new application features (play and tune a concept "online")

In fact, StepTalk is more than a scripting framework with an illusion of a single objective environment between objects of scriptable servers or applications. StepTalk, when combined with the dynamism that the Objective-C language provides, goes way beyond mere scripting.

The framework is language independent, that means that languages are provided as modules. The default scripting language in StepTalk is Smalltalk. Reason for the choice was that the Smalltalk is very simple language and it is easy to learn. There are just a few basic principles that the user has to know to be able to use the language and it is close to the natural language. Smalltalk uses a similar approach to that of Objective-C, the language used in GNUstep itself.

1.1 Of Scripting Languages and Language Engines

Scripting language - a communication tool for of telling objects on a computer what to do. You may ask: "What objects?" and I would answer: "Objects you see on the screen or objects you know are there."

– "How does the computer know, what objects I mean?"
– "You name them."

– "Will the computer understant if I would like to do something with object I call 'document'?"
– "Usualy yes - if the developer of the application would name the object you can think of as of 'document' by the same name"

– "If we both agree on the name, then I can just use it in my script, right?"
– "Yes. "

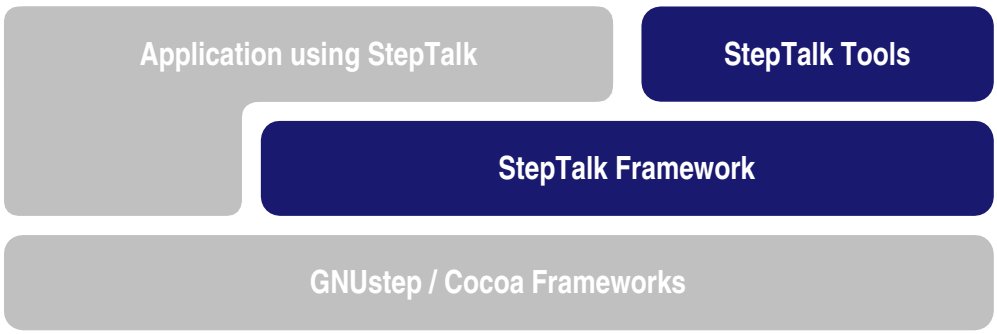
Now we see, that the language and scripting is about two main things: communication system (tool) and objects the tool is working with. In other words, the system is put into a small world called scripting context. It is the same with our natural language where we are allways in some context: office, restaurant, opera or disco. If I am in context 'office' and I will talk about object named 'table', then it would be obvious what table I mean by default - it would be the office table, not the table down in the cantine. The context links object names to real entities.

The language is represented by an engine (interpreter) that interprets scripts. As interpretation of all scripts is done in a context, the engine should take that fact into account. How? Primarily by consulting the context for object names. That means that when an object name (also can be called identifier, variable, constant...) is found in a script, the engine should bind it to real object through the contex.

To sum it up: there are two sides where context is considered and they are an application and a language. The application is context provider and the language is context user. While the application is responsible for giving object names that a user would expect to be, the language is responsible for resolving names in scripts to the same objects that applocation defines.

1.2 Architecture

The StepTalk consists of a framework, set of tools, language bundles and other modules. The framework is the core which binds all componentes together.



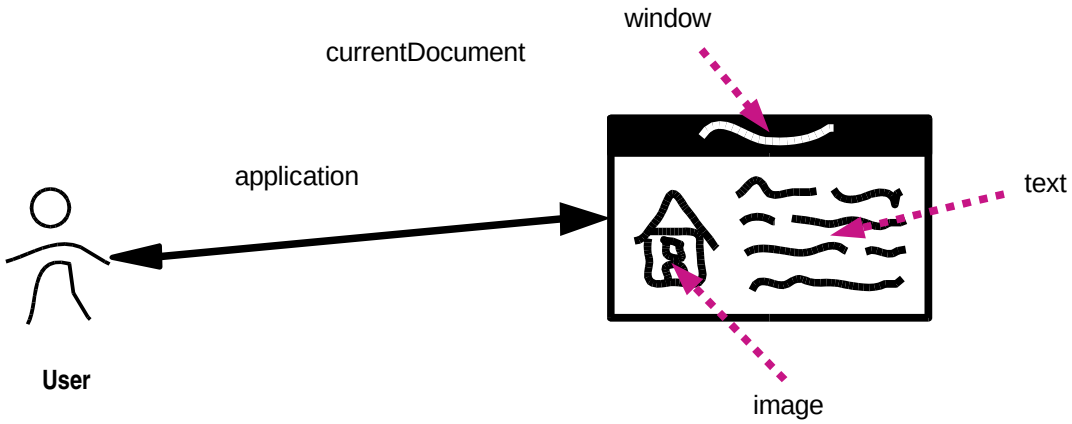
Drawing 1 Architecture

2 Scripting how-to

2.1 Scripting Session

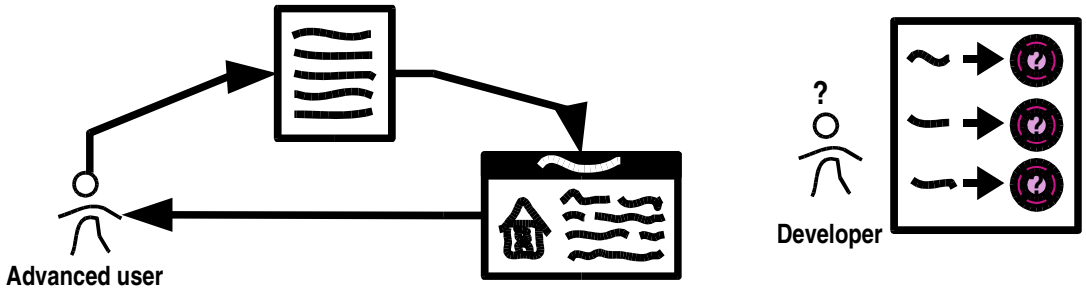
Before scripting

While using an application, the user can think of many objects that he sees. For example he can think of the current document being edited, active window, application or an image in the document.



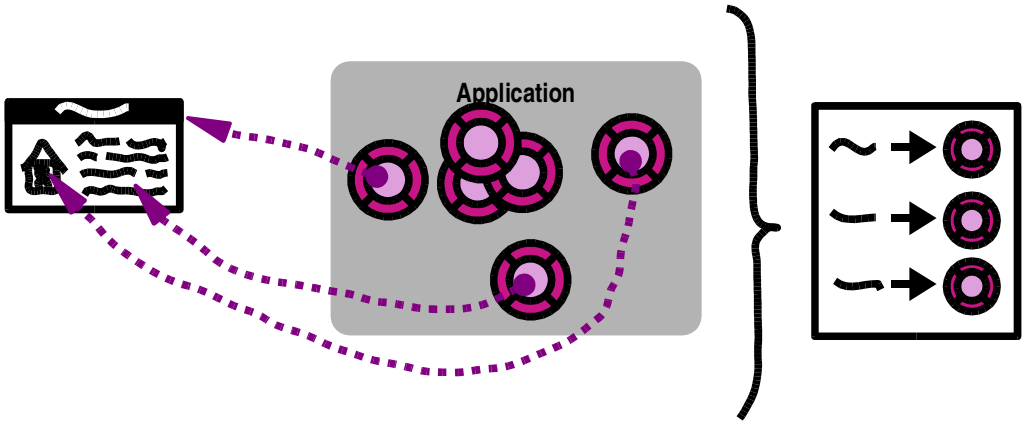
Drawing 2... user can think of many objects that he sees.

Advanced user would like to talk to those objects, automate several tasks or create complex tasks that would take lots of time if done manually. Therefore the developer has to map entities, that user needs to handle, to real objects inside the application.



Drawing 3Advanced user and developer

Each user-visible entity is somehow represented inside the application. Let us assume that each entity on the screen is a presentation of an object in the application. The developer has to prepare a mapping between words and objects inside a given context.



Drawing 4Mapping etween words and objects inside a given context

The developer prepares objects that would be available for scripting, therefore available for hanling by the user through scripts. He creates an environment (conversation context). This context will let the computer resolve object names. In other words, computer will know, what is meant by words *document*, *application* or *selectedObject*.

Creating environment

Environment, represented by the class STEnvironment, is a context that defines what is behind object names. User can create various environments, one for whole application, one for each document, etc. In the environment objects and classes from frameworks or modules can be included too. Also object finders can be associated for finding unknown objects or distant objects. Preparing a scripting environment is simple:

```
#import <StepTalk/STEnvironment.h>

STEnvironment *environment;

environment = [STEnvironment
                        environmentWithDefaultDescription]
```

or

```
description = [STEnvironmentDescription
                                  descriptionWithName:name]
environment = [STEnvironment
                    environmentWithDescription:description];
```

To register objects in the environment one should use *setObject:forName:* method:

```
[environment setObject:NSApp forName:@"Application"];
```

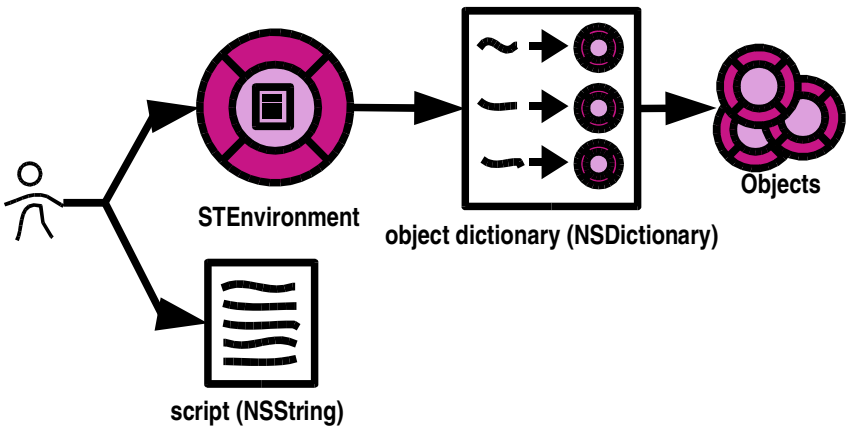
Setting other objects is similar.

Providing script

An application or a tool should have a ways how user can provide a script. It is up to the developer and user requirements. Here are few suggestions:

- text editor window with a button for execution
- open panel for loading a script to be executed
- stored scripts in bundels or frameworks that are used as extensions
- command-line provided scripts
- standard input stream

The application or tool developer does not have to forget that before he passes the script to the framework he has to know the language of the script. The language should be either specified by the user or can be determined from the file extension.



Drawing 5Required preparation before scripting

What is a script? It is a small program or set of instructions in user's preferred language understood by StepTalk. User should know what language the script is written in as there is no way how to guess the language from the script itself at the moment. The script can reference objects by their name. Referencing an object by its name means mentioning the object in the script. Objects are then resolved according to the associated execution context (environment) provided by the application developer. Technically a script is a NSString object at the moment.

Conversation

At this moment we should have prepared the script and the execution context or the environment. What we would like to do is to talk to objects, ut before that we have to prepare a conversation and make sure, that the conversation context (environment) is known. That means that the conversation object is connected to appropriate environment object.

STConversation is main object for “talking to objects by scripts”. It should be associated with a context (environment). For each execution an engine is created according to a script language. User can have as many conversations as he likes.

```
#import <StepTalk/STConversation.h>

conversation = [STConversation
                  conversationWithEnvironment:environment
                                     language: @"Smalltalk"];
```

Interpreting the script

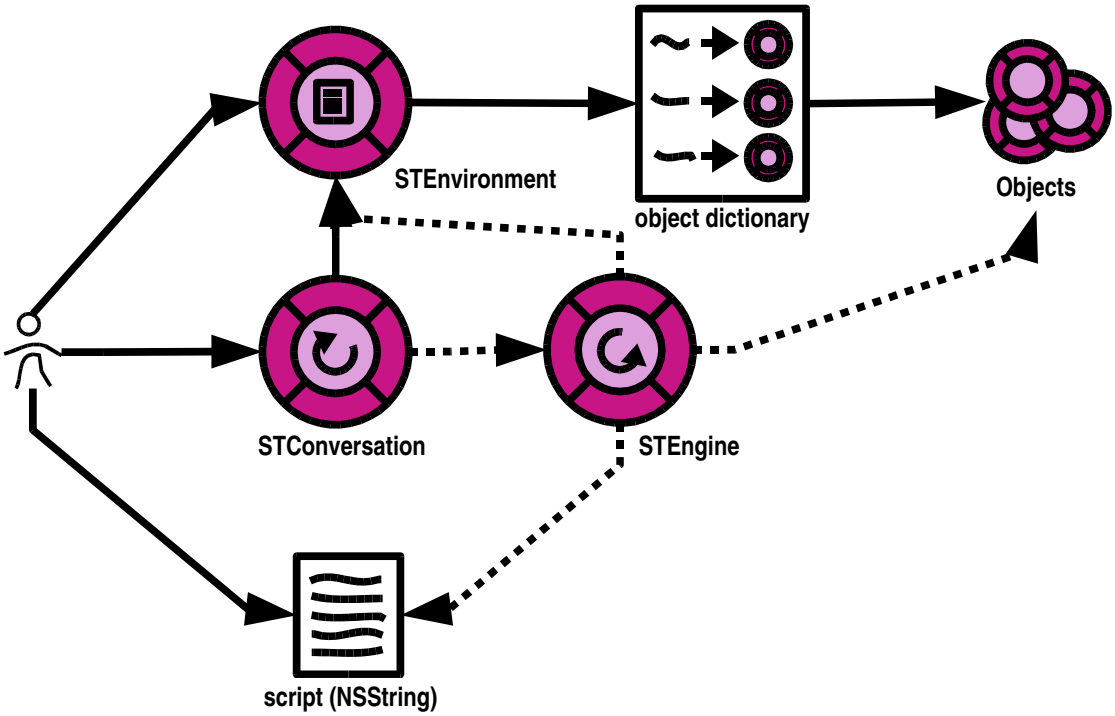
Now when you have opened a conversation in an environment you can run the script string. It is better to guard the execution with an exception handler:

```
id result;

NS_DURING
    [conversation interpretScript:string];
    result = [conversation result];
NS_HANDLER
    /* handle an exception */
NS_ENDHANDLER
```

What happens? An engine is created or reused for a given language. There is different engine for Smalltalk language or Ruby language, for example. Engine is set up to operate inside the conversation context (environment) and then the script is interpreted by the engine. Engine exists temporarily while code execution. Life of STEngine is undefined and depends on STConversation implementation. That means that for each execution new engine is created or they can be recycled.

While interpretation, each object referenced by a name is resolved through context object dictionary. What is ehind each object name and how the names are connected to objects depends on the language itself. The engine is provided with the object dictionary and it is up to the engine how the dictionary will be handled.



Drawing 6Objects involved in the script interpretation

Talking more

You can execute as many scripts as you like using the conversation. To change a language of script conversation, just use:

```
[conversation setLanguage:@"AnotherLanguage"]
```

To get names of all available languages:

```
STLanguageManager *languageManager;
NSArray           *languages;
NSString          *defaultLanguage;

languageManager = [STLanguageManager defaultManager];

languages = [languageManager availableLanguages];
defaultLanguage = [languageManager defaultLanguage];
```

You can then use the received list of languages in a pop-up button of your application for language selection.

2.2 Language bundles

Scripting language bundle is, well, a bundle that offers a language for StepTalk. The bundle provides mechanisms for understanding the language and interpreting it. Core of a language bundle is an engine and each language or language dialect should have its own engine that inteprets scripts.

The engine should take into account a context, where the script will be executed. After the execution, the engine should provide some feedback, for example a result object or it should throw an exception when something went wrong.

Few steps for creating a custom language bundle:

1. take a language template from StepTalk sources or create one of your own
2. create your own language engine as a subclass of the STEngine
3. implement method executeCode:inEnvironment:
4. provide an information list about the bundle

Template for a language bundle can be found in the StepTalk sources.

Preparation

Any bundle containing language engines should advertise its contents in the bundle's information property list. The information is contained in a dictionary named StepTalkLanguages where keys are language names and objects are dictionaries containing information about the languages. Here is an example:

```
{
    /* Array of file types of your language scripts */
    StepTalkLanguages = {
        MyLanguage = {
                FileTypes = ("mylang");
                EngineClass = "MyLanguageEngine";
            }
    }
}
```

There are following keys that define the language: *FileTypes* – array of file extensions for scripts written in the language, *EngineClass* – name of a class that can interpret scripts in the language.

Implementation

Header file for a custom language should look like this:

```
#import <StepTalk/STEngine.h>

@interface MyLanguageEngine:STEngine
{
    /* define engine variables here */
}
@end
```

And the implementation:

```
#import "MyLanguageEngine.h"

@implementation MyLanguageEngine
- (id)interpretScript:(NSString *)aScript
            inContext:(STContext *)context
{
    id retval = nil;

    /* interpret aScript while using the context for
       object name resolving and other context
      related things */

    /* retval should contan the return value from the
       interpreter */
    return retval;
}
@end
```

Interpreting in context

To get a named object:

```
NSString *name; /* name from the script */
id        object;

object = [context objectForName:name];
```

The context provides more than simple name lookup. Language developer can (and perhaps should) use the the selector translation finctionality. This functionality can be used for kind of security for the script execution. Before sending a message to an object it good to check whether we are allowed to send it and how the selector is translated:

```
NSString *selector; /* requested selector to be sent */
```

```
id        anObject; /* the receiver */
NSString *newSelectorString;

newSelectorString = [context translateSelector:selector
                                     forReceiver:anObject];
```

The *newSelectorString* will containin translated selector or *nil* if the selector should not be sent (because it is fobidden by the environment, for example). Moreover you should use this method if you want to use restrictions of selectors of the scripting envornment. See STEnvironment and STEnvironmentDescription documentation or headers for more information.

2.3 Security

StepTalk offers several mechanisms for script execution security:

- object availability throught scripting environment
- selector restrictions

Scripts can access only objects that are available in the scripting environment. Therefore it is reccomended to name only objects that can be considered as secure. Objects that are not named are not directly accessible. Object naming is done by STContext/STEnvironment method -setName:forObject:.

Another way how to get access to restricted objects or how to execute restricted method is by sending restricted messages to objects. StepTalk offers a mechanism for denying selectors on certain classes. If a selector is denied and message is sent, then an exception is raised and the method is not executed.

Restrictions can be done either by allowing all selectors in a class and denying some of them or denying all selectors in a class and allowing only some of them. Both methods have their advantages and disadvantages as well as their particular usages.

Selector restrictions are listed in environment descriptions. The scripting engine has to consult the execution context for the restrictions. Beware that if the engine does not do the checks, then the restrictions are ignored! Smalltalk engine checks for the restrictions.

For testing purposes and while StepTalk is in its prototype stage, there is an option in the STEnvironment for enabling 'full scripting'. That means that one can configure the environment to ignore the selector restriction checks. To be sure that the chcecks are done use:

```
[environment setFullScriptingEnabled:NO];
```

Note to the engine developers: to do selector restriction checks use STEnvironment method -translateSelector:forReceiver: