

# Multilingual Flash® Applications

Presented by Robert Taylor  
Flash Extensions, LLC  
flashextensions.com

Notes:

## The Session

Companies are constantly seeking to gain an edge over their competition. A common bullet point many products emphasize (if they have it) is localization. In this session, you will learn how to architect Flash® applications to support multiple languages. We will discuss how to build flexible language structures that can be read from files or databases. Also learn how to establish events in order to change language content without reloading your application or refreshing the browser. We will further discuss techniques for applying visual designs around different languages. Finally, learn techniques to implementing RTL (Right-to-Left) support for languages such as Arabic and Hebrew.

## Who Should Attend?

If you're looking to broaden your audience by extending your applications to support multiple languages, if you would like to personalize content based off of a user's profile, or would like to display different design elements based off of regional settings, attending this session will provide you with the knowledge to accomplish these tasks.

## General Planning & Architecture

*"If you fail to plan, you plan to fail..."* – Tariq Siddique

For the past few years online apps have grown in sophistication as users have grown in experience. *Usability* is becoming as important as *functionality*. One feature that compliments both elements is the ability to support multiple languages. We have all been to websites that are published in a language other than our own. Generally, the stay is short because there is only so much that can be gained. However users are starting to explore beyond their regional scope. This is in part because of blogs and other sites that encourage exploration on the web. Implementing multilingual support will not only increase user experience, it raises expectations for the next generation of online apps. The challenge many of us now face as developers, is how to integrate language support into our application's architecture. Below is a composite of questions to address at the beginning of each project:

*"multilingual support...raises expectations for the next generation of online apps."*

- What is the default language?
- What other languages will be supported?
- Do the languages differ greatly in string size?
- Can the user switch languages at runtime?
- How will different sections of the application be notified when the language changes? How will these sections receive their strings?
- How can we insert dynamic content into static strings?
- Are there ways to optimize retrieving language data?
- How can we easily switch between reading local files to reading data from a server?
- Are there other things we should be aware of when building localized applications?

We will be addressing these questions along with others that are often overlooked. We will also discuss the different language types, the challenges and solutions for overcoming their limitations.

## Choosing a Default Language

Though English is the most common language for applications, it is not necessarily

# Multilingual Flash® Applications

your application's default language. You will want to select your default language based off of your user-base. It is also possible to regionalize the application and select a default language based of the area in which the application is being viewed. You could also have the user select the language before the application launches. This not only sets up the user for a better experience, it could possibly save an extra call for retrieving string data that may never be used.

## Supported Languages

Combined, your application and audience will determine the languages you need to support. I have found the most commonly supported languages are English, Spanish, French, German, and Japanese. If it isn't necessary for your application to support multiple languages, don't implement it just because you can. The same is true for any language. If a Japanese audience is never going to view your application, don't integrate the language just for a small percentage that may. Each language you support will likely add complexity and design changes to your interface. Some language strings differ in word count; others may require an embedded font to display properly. Some languages require an IME (Input Method Editor) for text entry. Each language you add will have to account for these possible dependencies.

*"If it isn't necessary for your application to support multiple languages, don't..."*

## Switching Languages at Runtime

One of the greatest benefits Flash® offers is its ability to dynamically change at runtime. The ability to switch languages at runtime without restarting the application is one that many other environments envy. In order to support "runtime switching", we need to have an environment that allows us to communicate globally back and forth to request and receive string data.

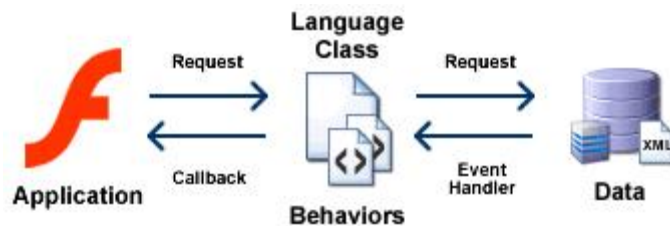


Figure 1 - Flow Diagram for String Requests

Figure 1 demonstrates how we might achieve this two-way communication. In this example there is base class called `Language`. Each section of our application enabled with language support will subscribe to `Language`. When the display language is switched all subscribers will be notified. Each subscriber then requests string data from `Language`. Once `Language` retrieves the data it performs a callback to the subscriber.

## Tag, You're It!

It doesn't seem like we can build applications without personalizing them anymore. Our application hasn't been narrowed in this regard just because we are using static strings. We can personalize our strings by using a tag-based system. Tags are used to predefine areas in our string that will later be replaced with dynamic data. It is simple, fast and straight forward. Our tagged strings may look something like this:

```
"Hello [firstname], you have [count] new emails in your inbox [daytime]."
```

Once we have replaced the tags, it will look like this:

```
"Hello John, you have 12 new emails in your inbox this morning."
```

# Multilingual Flash® Applications

Sentence structure will change depending on the language, Tagging allows us to customize the string without creating an awkward sentence structure.

## Tips on Optimizations

- Perform Requests on Visible Items Only

Depending on the complexity of your application, you may have several sections with language support. With the architecture introduced in Figure 1, we discussed how these “subscribers” are notified when the language has changed. One way to balance the load from too many requests is to only request string data if a section is visible. It is possible that a section of your application may never be viewed during a session. This will reduce unnecessary requests from occurring and will provide a better experience to your users. When a section does become visible it can then make the appropriate request to update its string data.

- Store Strings in Memory

Because it is unlikely the string data will change often, there are some things we can do to optimize the request flow in our application. The first applies to both local and remote data calls. Once the application has received data, store it for future requests. If a user switches between languages more than once the app can prevent unnecessary data retrieval by accessing the stored string data. If your application has several language dependent sections, this can really speed up the process.

- Cache Queries

Though this is a common technique for search queries, I still thought I should mention it. If you are retrieving string data from a database, you may want to cache these queries to increase performance. Once the data has been cached, it is the same as retrieving them from a file. The cache can be set to expire so if changes are made, they will be reflected with after the expiration.

## Strategy Pattern

The key to building a stable multilingual application is flexibility. In Figure 1, the base class, `Language`, implements the Strategy Pattern and uses supporting classes called “behaviors” to determine how to retrieve and interpret string data. By using the Strategy Pattern, we change the manner in which we get data without the need to rebuild our `Language` class for each project. This implementation also allows us to change the way in which we retrieve data at runtime.

*"The Strategy Pattern...allows us to change the way in which we retrieve data at runtime."*

## Language Type Considerations

There are three language types our applications may support: LTR (Left-to-Right) languages, Asian languages, and RTL (Right-to-Left) languages. Each has unique challenges to consider when developing a Flash® application. Some challenges are confined in the language. Others are introduced when the application supports more than one language type.

### Left-To-Right (LTR) Languages

Of all the types, LTR languages are the easiest to develop for. Latin symbols are generally supported by the most popular fonts. The single most common problem occurs with layout. Oftentimes, the translations differ in length or word count, so we must be considerate when introducing another LTR language into our application's design.

# Multilingual Flash® Applications

---

## Asian Languages

Though Asian languages are displayed Left-to-Right they are unique enough to warrant being placed in their own category. If the application is going to support Asian languages, we should check to see if a compatible font exists on the user's machine. Typically, these fonts are not installed by default. This could result in the user seeing "□□□□" instead of the appropriate string because the system cannot find a compatible device font. Additionally, Flash® cannot render device fonts with anti-alias support without embedding the font. This sometimes leads to unreadable text if the font size is too small. Though embedding the font into your application may fix this problem, the increased size from the font (in some cases embedded font may increase the SWF by several megs) often leads to unacceptable file sizes.

## Right-To-Left (RTL) Languages

RTL support in Flash® is a misnomer because there really isn't any. When it comes to supporting languages such as Arabic and Hebrew, Flash developers are required to get a little creative. Like Asian languages, the user is required to have the font installed on their machine in order to correctly display the strings. Additionally, features such as word-wrap often break because the font is rendered differently from other device fonts.

## Supporting Multiple Language Types in a Single Application

As soon as you want to support multiple language types, you need to consider the challenges each language type carries. The biggest challenge you may face is creating localized applications with a small footprint.

## Input Method Editor (IME) Support

Most applications will require the user to type into some form of entry field. Double-byte characters as seen in Asian and RTL languages use what is known as an IME to submit complex strings from the keyboard into these fields. Though Asian languages work pretty well with IMEs, RTL languages encounter a number of problems. Some of the challenges you may encounter are text selection, cursor navigation, and mixed language types within the IME.

Once we know the possible challenges we could face when developing our localized application, we can now move forward to architecting solutions for these issues as well as the means to load and manage the different languages we might support.

## Custom Solutions to Common Problems

In order to overcome some of the barriers introduced through multilingual support, you might have to rely on some of your own creativity and programming skills. Fortunately, you won't have to depend solely on it. Let's go over a few items that will help you implement fluid language enabled systems.

### Fontastic

Fontastic is a solution to dynamically load embedded fonts into your Rich Internet Application without requiring you to have defined an instance or shared resource in the application's library. It can allow users to view any font, including double-byte characters found in Asian languages. The advantage that Fontastic gives you is the font can be loaded upon request or it can be shared across other flash applications, and reduce the overall file size of your application through reuse.

*"Fontastic is a solution to dynamically load embedded fonts...including double-byte characters"*

# Multilingual Flash® Applications

## Arabic Text Viewer

Arabic Text Viewer is a special control for displaying Arabic in Flash. Arabic is a serious pain to display in Flash®. It has several shortcomings that prevent it from displaying properly such as improper word-wrapping of symbols and mixed language support (i.e. English and Arabic within the same text box). Flash® isn't really to blame for these problems. Other applications and environments are susceptible to the same problems. The Arabic Text Viewer helps to overcome some of these problems.

## Other Solutions

There are others who have developed solutions to other common language limitations. Rather than discussing them here they will be available at the URL listed in the Conclusion. Links will increase or be modified as new information is found.

## Beyond Words

I love solutions that can resolve more than one problem at a time. So far we have focused on solutions for text-related issues regarding language support. But our model allows us to go above and beyond text.

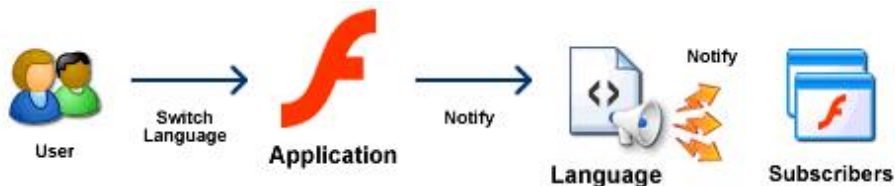


Figure 2 - Broadcasting Switch to Subscribers

Figure 2 demonstrates the user switching languages and then all subscribers getting notified. Once a subscriber is notified, it knows the language that is currently being represented. The subscriber doesn't have to limit this information to strings. For more complex layouts, you may have created a different SWF or MovieClip to represent the language or style. With this mechanism in place, it is really up to you to do what you want. In fact, with this method, you may not be dealing with languages at all. It may be the means for you to change themes or content. Creativity will need to pop in at this moment to fill in the \_\_\_\_\_. Customizing and personalizing the application results in a better user experience, which in the end, is the whole purpose to this application anyway, right?

## Conclusion

Implementing languages can be very rewarding and will not only please your clients but those using your applications. Through careful preparation and practice, you will soon find yourself easily implementing localized projects. For more information regarding this session, go to <http://www.flashextensions.com/blog/>. Session notes and other materials will become available after conference.