

National University of Singapore
School of Computing
CS3216: Software Development on Evolving Platforms
Google Wave Extension Assignment

:

General Overview

According to Google, Wave is "what email would look like if it is invented today". Also according to Google, there are serious drawbacks with working on a project using emails [0]:

1. **Propagation of email leads to multiple copies and versions.** Different users will be getting different copies of messages overtime and some may not be updated on the latest development.
2. **Rich content such as maps, photo slideshows, and video clips cannot be embedded in the body of an email.** The only way to do it is through the email's attachment feature.
3. **Difficult to reply to a particular section of an email.** To reply to a certain section of an email, one needs to look for the section and quote it manually.
4. **Difficult to address to specific people in a group email.** To do that, users have to manually delete unwanted recipients from the mailing list.

In view of these problems, Google Wave is designed as a better tool to facilitate such collaborative processes. Using Google Wave, the users will be able to

1. **Eliminate the problem of receiving multiple copies of outdated messages.** There will only be a single copy of the message hosted online and any changes will be updated real time. There is also a playback function that allows users to trace how the message has been modified over time by different users.
2. **Allows rich content to be embedded within the message.**
3. **Reply to a section of a message inline.** To reply to a specific section of the message, the user can just click on it and type in his or her reply.
4. **Reply to specific people within a group.** Similar to forums, the users can start private threads within a group message.

Currently, Wave is in an invite-only preview stage. This preview release was launched in September 2009. In the coming months, it is anticipated that the number of users will be growing exponentially. If you can understand the users' needs, you can leverage on this opportunity as pioneer developers to make a difference!

To better understand Google Wave, you may wish to check out the following introductory video (about 1 hour long): http://youtu.be/v_UyVmITiYQ

Acknowledgement

We would like to express our gratitude to Google for their help and support in making this assignment possible, especially *Stephanie Liu*, who helped us obtain the sandbox accounts and *Pamela Fox*, who provided us with invaluable feedback on the assignment.

Grading and Admin

This assignment can be done in groups of three or four students. If you are unable to find yourself a group, you will be randomly assigned to one.

We will not be providing step-by-step instructions. Instead, we will only list the key concepts you need to master (also known as "milestones"). We will also provide some related tips, references and a little bit of help to get you started. These milestones constitute 70% of the assignment's grade.

Like the previous assignment, if you find that some of the proposed milestones do not make sense for the application you intend to build, you can petition to replace them with some other deliverables. You are to explain why we should agree to your petition, and submit your petition at least one week before the assignment is due. Your petition is subjected to approval.

While the milestones may be easy to meet, simply meeting them will not give you the maximum credit. We ask for quality submissions, not run-of-the-mill work.

To score the coveted remaining 30%, we would like to see how you can put your creativity to the test and explore all pores of your originality tissues. We choose not to restrict your potential by insisting on any particular sequence. We strongly recommend that you blow us away with your creativity.

Please do not hesitate to approach the friendly CS3216 staff if you need further assistance. If you have questions with Google Wave, please post your questions in the IVLE Discussion Forum or better create a Wave and discuss it with a Wave! Whatever works. :-)

Objectives

In this assignment you are required to deploy a Google Wave application. Your goal is to demonstrate that you can implement a Wave gadget, a Wave robot, or both.

While we believe (and Google believes too) that the most sophisticated extensions will be combinations of gadgets and robots, and we did consider making every group do both. However, we decided that we did not want to be prescriptive so as to provide you with maximum flexibility and minimal restrictions on what you can do. Extra credit will be awarded to groups who incorporate both in a seamless way to do something cool.

Remember, your goal is not to do a lot of work. Your goal is to use this opportunity to "make a difference". If you can make a difference by just doing a little bit of work, that's fine with us.

Before you begin, do spend some time understanding the requirements for the assignment. Also, try to get some sleep before you start on this assignment. To help you avoid losing sleep, we have included a detailed grading scheme at the end of this handout.

Introduction

Welcome! This assignment comprises of 2 weeks of intensive learning that provides you with another opportunity to express your creativity.

To enrich the users' experience, the Wave platform is open to external developers. These new features and functionalities that external developers develop are called *Wave extensions*. Like Facebook applications, Wave users can add these extensions by clicking on an installation button.

There are two types of Wave extensions [0]:

1. **Robots.** Robots are automated participants within a wave which can talk to users and interact with waves. It can be used to provide information such as stock quotes from external sources.
2. **Gadgets.** A gadget is an application where users can join to interact with each other. You can think of the gadget as being similar to iGoogle gadgets or Facebook applications.

To be even more accurate, an extension does not need to involve only a gadget or robot [0]. It can also involve both a robot and a gadget [0].

You can choose to develop either a robot, a gadget, or both. If you choose to build a wave gadget, you need to complete seven mandatory milestones (4-10). If you choose to build a robot, you are expected to complete three compulsory milestones (11-13). In addition, there are eight milestones (milestones 0, 1, 2, 3, 14, 15, 16 and 17) that are common to both paths of development. Out of these eight milestones, milestone 16 is optional and milestones 2 and 3 are not graded.

Before you start, please take a moment to check out the Extension Design Principles [0] (and see milestone 14).

Reminder:

You are given a choice as to what you want to build (either a gadget or a robot, or both :-P). To ensure that you will make the right choice, please take some time to read through the entire assignment before thinking about what you want to develop. You are also welcome to do some simple experiments to understand what Wave can and cannot do.

Thinking hard about your Wave Application

No charge for awesomeness!

– Kungfu Panda

Since you are among the first Google Wave extension developers in the world, you are likely to be excited about sharing what you are doing with your friends. Mention “Wave developer” to your friends, and they will be impressed.

As exciting as developing on a new platform may seem, it also has problems associated with being new. Why would you want to develop on the Wave platform instead of another more established platforms, such as Facebook?

Building a killer Wave application requires more than just technical skills. In CS3216, we expect you to think very hard about what you're trying to do. You should not be building a Wave application just because you need to submit this piece of homework.

You should choose an application that truly exploits the potential of the Wave platform. For example, you should have a good answer to the question of why developing on other platforms is not the best solution to fulfill your application's objectives.

In CS3216 (and life in general), execution matters. Identifying a good idea is the first step, deciding on which path of development to take, that ensures maximum success for your application, is the next. Deciding whether to build a gadget or a robot (or both) is crucial in your execution. Thus, we expect you to come up with a good reason for it.

Milestone 0:	Describe your application and explain how you have exploited the Wave platform to achieve your application's objectives, i.e. why is Wave a more suitable platform than other platforms like Facebook to develop your chosen application? You can choose to develop either a gadget and/or a robot. Explain your choice.
---------------------	--

Unlike the Facebook application you have developed in the first assignment, your Wave application does not have a ready-made social network to leverage on. It is no good to have a killer app that nobody uses. Hence, you will also be expected to think a little harder about how you plan to "market" your app to potential users. You must identify your target users, determine the relevance of your application to them (i.e. why should they care about your application) and explain how you plan to reach out and persuade them to use your application.

In order to promote the use of your application, good marketing strategies are crucial in raising awareness of your application among the targeted users. After introducing potential users to your application, how would you try to persuade them to continue using the application, and perhaps, even share or introduce it to others? What value do the users derive from using your application?

Ideally, you should also think of ways to provide motivation for users to promote your application to their friends. The application should be designed in such a way that while individual user may derive some values using your application, it is in their interest to promote your application as more users will make your application more valuable to them.

Your efforts in promoting your app should also take into consideration that Wave is currently in its infancy and that users get their wave account only via invitation. Working under this restriction, how would you ensure that your target users will be attracted to your app?

It is also reasonable to expect that the number of Wave users will grow rapidly in the foreseeable future. Given this trend, how would you take advantage of the growth to increase the number of people using your app?

Milestone 1:	Describe your target users. Given that Wave is a new platform, explain how you plan to promote your application to attract your target users.
---------------------	---

Before you start doing anything with Google Wave, you need an account, which you should already have at this point. There are 2 types of Wave accounts: regular and sandbox accounts. They are almost identical, except that the sandbox account is meant for developers. The sandbox account will have earlier access to new features and APIs, which will only work in the Sandbox environment. The Sandbox environment also has a debug interface which is not available in the regular Wave environment. You should develop an extension in the Sandbox environment first and move it to the regular Wave environment only when you're done. If you decide to develop an application that uses Sandbox-environment-specific APIs, you don't need to move to the regular Wave environment. Just indicate this clearly in the submission README file.

Milestone 2: Set up your Google Wave and sandbox account. Update the profiles with your real name (so that we can identify you) and upload a picture (*Not graded*).

After a discussion with your groupmates, you should have an idea of what you want to build. Now, it is time to pick a name for your Wave extension. It is also a good idea to write a short description about what your extension can do. Just a short paragraph will do (3-5 lines).

Milestone 3: Pick a good name for your Google Wave extension, along with a description for your Google Wave extension. (*Not graded*).

Wave Gadget

This section is not relevant if you are building a Wave Robot, but you are encouraged to read through and understand what a gadget can do. After all, if you decided to change your mind with regards to the kind of extension you would like to build, it is always better to change your mind early.


Getting Started

The following is a simple "Hello World" gadget. This should take you less than 5 minutes to complete. Create a file name `hello.xml` with the following content:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
  <ModulePrefs title="Hello Wave">
    <Require feature="wave" />
  </ModulePrefs>
  <Content type="html">
    <![CDATA[
      Hello, Wave!
    ]]>
  </Content>
</Module>
```

or download the file from: <http://gadget-doc-examples.googlecode.com/svn/trunk/wave/hello.xml>

Upload the newly-created file to your webserver (this could be any publicly accessible webserver, i.e. your SoC unix hosting works for this. Each group will be provided with more Amazon credits for this assignment, so use it wisely.

Login to your Wave account, create a new Wave and click on this button  (only available if you're in edit mode).

Go to the URL you have noted down, you should see the text "Hello, Wave!" in your wave. Congratulations, you have just built your first gadget :-).

Take a look at this segment of code again:

```
<![CDATA[
    Hello, Wave!
]]>
```

It contains the bulk of the gadget, including your gadget's logic and presentation. Next, we will run through how you can add these elements into the gadget.

Building your gadget

The *Hello World* gadget only has 1 line of text and is quite boring. In this section, you will learn how to use CSS and JavaScript to make your gadget look better.

```
<![CDATA[
    Your code goes here
]]>
```

Any code inside `<![CDATA[]]>` is treated as the content of the body tag in HTML file. It can contain CSS and JavaScript, or refer to them.

Presentation

Your team (or maybe just your user interface designer) should spend some time designing a good UI. A good UI is the key factor that attracts users to your gadget. Although the functionality of your gadget is important, the way that it provides the functionality is just as important. A gadget that is difficult to use won't be used. Period. It won't matter how technically superior your gadget is or what functionality it provides. If your users don't like it, they simply won't use it. Seriously, do spend some time getting it right. In most cases, you'll know immediately if your UI makes or breaks it. It's common sense(!).

According to web standards, you should use CSS to style your webpage. Since the Wave gadget is just another webpage embedded in a Wave conversation, the same principles applies here. CSS stands for Cascading Style Sheets, a language to determine the formatting and layout of a web page. It provides a way to separate content from presentation, thus helping us to organize the code in a much more efficient manner.

All your styling should be contained within a CSS file (you can link an external file to your gadget) or clearly defined at the beginning of your gadget. It is very bad practice to mix CSS into HTML code.

Milestone 4: Style different UI components within the gadget using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application.

After you have created a fancy UI, you need to teach your gadget what to do.

Behaviour

Wave gadgets use JavaScript to define its logic. Besides pure JavaScript functions, it has access to 2 sets of APIs:

- Google Gadget API: <http://code.google.com/apis/gadgets/docs/reference/>
- Wave Gadget API: <http://code.google.com/apis/wave/extensions/gadgets/reference.html>

Google Gadget API allows you to build gadgets that can run on multiple sites, such as iGoogle, Orkut (Google's social networking platform), Google Wave or your own site. You can go to <http://www.google.com/ig> and start adding stuff to see what Google Gadget is about.

A gadget on iGoogle will most likely run fine on Google Wave, except that it would not be able to take advantage of many new features offered by Wave (such as real time synchronization and multi-user environment). By using Wave Gadget API, you can make use of these features to for your gadget to interact with the Wave conversation/participants.

Wave gadget works with callback functions. A callback function is a function that is passed to another function (in the form of a pointer to the callback function) so that the second function can call it. This is simply of way of making the second function more flexible without the second function needing to know a lot of stuff. By passing different callback functions, you can get different behaviours. A function can take multiple callback functions as a parameter and execute accordingly.

Here are some interesting functions that take another function pointer as parameters:

Google Gadget API

`gadgets.util.registerOnLoadHandler`: call the callback function right after the gadget gets loaded.

Wave Gadget API

`setParticipantCallback`: call the callback function every time the Wave's participant list changes. This is ideal for gadgets that need to track the participants in a Wave conversation.

`setStateCallback`: call the callback function every time the Wave's state (mentioned later) changes.

To start, we need to define which function to call when the gadget is first loaded. From there, we will continue adding more callback function.

Let's try to pop up a message when we load the gadget. Copy this chunk of code and put it on your server:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
  <ModulePrefs title="Hello Wave" width="400" height="120">
    <Require feature="wave" />
  </ModulePrefs>
  <Content type="html">
    <![CDATA[

      <script>
        function init() {
          alert('This is a pop up');
        }
        gadgets.util.registerOnLoadHandler(init);
      </script>
    ]]>
  </Content>
</Module>
```

`gadgets.util.registerOnLoadHandler` defines that the function `init` will be called when you first load the gadget. The `init` function simply pops up a message in your browser.

By default, `wave` sets your gadget height at about 1 text-line tall. The width, on the other hand, is dynamic as you resize your window. That means if you don't specify the height in your gadget (see code above), your gadget will display only the first line. `Wave` omits the scrollbar too, so users will not be able to recognize that there's more information below. It is recommended that you set the width and height of your gadget correctly to avoid this situation.

If your gadget requires dynamic height, you could use the dynamic height feature in Google gadget API. More details are available here: http://code.google.com/apis/gadgets/docs/ui.html#Dyn_Height

Store your data

Google Wave gadgets typically do not use a relational database to store the shared state information. Instead, key-value state is used. In a key-value state, there is no row or column. Every value is associated with a key name. The only way to access a value is to know its name.

Before you store any information in the gadget's state, note that the state is only available within the gadget in a Wave conversation (e.g. a gadget cannot access state of another gadget in the same/different Wave conversation). Furthermore, the state object is *shared* among every participants within a Wave conversation. Any participant can make any change to the state any time he or she wants. You should remember the following when you develop your gadget:

- Only expose information that is meaningful to everyone on the wave.
- Do not expose any private information of a user. If you need to, let them know before they enter such information. Users' privacy is important. Violation of this rule will lose your users' trust in no time.
- There could be instances where many users try to modify a state at the same time.

The state object is accessible via Google Wave gadget API. Some basic APIs that might be of interest: `submitValue` (create/update one key-value pair), `submitDelta` (create/update the state delta), `get` (retrieve a value).

Callback functions

As mentioned, Wave gadgets work by registering callback functions. Changes to a wave's state is an important event that your application may need to response to. Using the API, it is possible to call a callback function every time the wave's state (*setStateCallback*) or the wave's participants (*setParticipantCallback*) changes. Once a callback function is called, it means your state is updated or participants list is changed. Your code inside the callback function will receive the latest change of your state and participant list. Therefore it is a good idea to put programming logic inside callback functions.

Milestone 5: Show 2-5 gadget API and wave gadget API calls in your Wave gadget that you consider to be the most interesting. Explain how these API are used to do interesting things in your gadget.

Milestone 6: Store some information in your gadget and modify/delete them (in an appropriate way).

Note: if you're stuck with the wave's state, Google has a nice guide to follow here <http://code.google.com/apis/wave/extensions/gadgets/guide.html#state>

It is a good idea to modify the state object when your UI component fires an event. Almost any interaction on any UI component fires up an JavaScript event. For example:

- On submitting a form.
- On clicking a button.
- On mouse down on a component

Note: You cannot submit a form in a wave conversation since your gadget is running within the wave environment only. You should bind an event handler to the submit button or the form itself to run your code on form submission. This would be identical to using a button and binding an event handler on clicking that button.

You will most likely want to capture user input via the UI. In most cases, it makes sense to update the state object after the user inputs something. Please do not add an event listener to your code just because we ask you to do so. Do remember that every single event that happens in your gadget contributes towards the user experience.

Milestone 7: Cite 2 or 3 examples where you use JavaScript to respond to events on UI components. Explain.

Wave comes with a set of useful built-in features, such as the realtime updating of state object and spell checking. This means that you can save yourself time from having to synchronize information among participants or implement a spell checking mechanism. While Google has solved some of the hardest problems with online collaboration, how you make use of these features is something that you will have to figure out. You should keep these features in mind when you're building your gadget and use them creatively. How much creativity is enough? We'll leave that to your imagination =).

Milestone 8: Explain how your gadget makes use of built-in wave-features (e.g. real time update, playback, etc.).

User experience

Another important part of your gadget is user experience. Please note that user experience (usually addressed as UX or UE) is different from user interface (UI). A good UI does not guarantee a good UX at all. Sometimes, a cool-looking UI can be a disaster because of poor UX.

User experience encompasses all aspects of the end-user's interaction with the gadget. The first requirement for a good user experience is that the gadget allows the users to do what he wants with minimal fuss. Next, comes simplicity and elegance which will make the gadget a joy to use. User experience goes far beyond giving user what they say they want, or providing a checklist of features. In order to achieve high-quality user experience in a gadget, there must be seamless integration of the gadget's functionality, interaction design and interface design.

User experience is not just the job of the UI designer. Just like a good UI, you will know if a gadget's UX makes or breaks. It is again, just common sense. Any team member can contribute to your UX design by using it and provide feedback and suggestions. Ask your friends to use it as well to gather more feedback and ideas.

Milestone 9: Describe 1-3 user interactions within the gadget and explain why those interactions help make the gadget better.

Google Analytics

Just like the Facebook application, you might be interested in the usage statistics of your gadget. While Facebook applications have access to Facebook Insight, which provides a lot more information about your application, the only mechanism you have in gadget is to embed Google Analytics (or other tracking mechanism) in your gadget.

However, do note that Google does not recommend the use of Google Analytics in gadgets, because Google Analytics will record every access a user makes to a wave conversation. Google's privacy policy for Wave states that a user should not reveal that he is on a wave until he is actually editing the wave. Google is working on Wave to make sure it complies with this privacy policy.

If you want to include Google Analytics in production gadgets, make sure that you inform the users about the consequence of adding the gadget (i.e. being tracked even if they are not editing a wave).

Milestone 10: Embed Google Analytics on all your pages and give us a screenshot of the report. (Optional)

Wave Robot

Like the Wave Gadget section, you are not required to read this section if you intend to build a Wave Gadget. However, we would recommend that you read it anyway. :-)

Definition. *A robot is an automated participant on a wave. A robot can read the contents of a wave, participates in it, modify the wave's contents, add or remove participants and create new blips and new waves. In short, a robot can perform many of the actions that any other participant can perform¹.*

Getting started

Currently Google Wave only supports robots hosted on Google App Engine². First you'll need to register for a Google App Engine account first.

After getting a Google App Engine account, you can create and deploy up to 10 applications with your account. You cannot change application name after you have created it, so pick the name wisely. You can see the list of your applications here: <https://appengine.google.com/>

You can use either Java or Python to develop your robot on Google App Engine.

Wave Robot API Library

- For Java:
 - Robot library: <http://code.google.com/p/wave-robot-java-client/downloads/list>
 - Robot API reference: <http://wave-robot-java-client.googlecode.com/svn/trunk/doc/index.html>
- For Python:
 - Robot library: <http://code.google.com/p/wave-robot-python-client/downloads/list>
 - Robot API reference: <http://wave-robot-python-client.googlecode.com/svn/trunk/pydocs/index.html>

You'll need Java 6 development kit or Python 2.5 or higher installed on your system. You can check your Java/Python version using the following commands:

- Java: `java -version`
- Python: `python --version`

¹<http://code.google.com/apis/wave/extensions/robots/#Introduction>

²<http://code.google.com/appengine/>

If you don't have the correct version, download either JDK 6 or Python 2.6.4 using the following links:

<http://java.sun.com/javase/downloads/index.jsp>

<http://www.python.org/download/>

Robot Identity

To register for a new application on Google App Engine, go to <https://appengine.google.com/> and follow the instructions.

After registering for your application, you'll be given an application id `application-id.appspot.com`. Your robot's address will be `application-id@appspot.com`. This is similar to your wave address `username@googlewave.com`. To add your robot into a wave is just like adding a new human participant.

Hello world!

Python

After you have downloaded your Python API Library, extract it and rename the folder to *waveapi*. Create another folder and move the *waveapi* folder to the newly created one.

We will start building our first robot. This robot will respond when you add it into a Wave conversation, or when it sees the participants list changes.

Create a file name *app.yaml* in your source directory (which means it is on the same level with *waveapi* folder) with the following content:

```
application: applicationname
version: 1
runtime: python
api_version: 1

handlers:
- url: /_wave/.*
  script: applicationname.py
- url: /assets
  static_dir: assets
```

Warning: Indentation is important in YAML, you need to make sure the indentation in your *app.yaml* file is correct.

YAML stand for **Y**AML **A**in't **M**arkup **L**anguage. It's a human-friendly data-serialization standard for all programming languages³. There are a few things you might want to take note in this simple example:

- *application*: the name of your application. This must be in lowercase and exactly like your appspot ID (Google App Engine use this application name to recognize your app).
- *version*: define the version of your robot. Google App Engine allows you to have at most 10 versions of the same app store on Google App Engine. Login to your Google App Engine, there's a menu call "Version" in Administration area. You will be able to change the default version for your app as well as delete some outdated versions.

³From <http://yaml.org/>

- *handlers*: specify URL pattern to serve static resources for application.

You can see this YAML file as a configuration file in your application. It tells the Wave client some information about your robot in order to work correctly.

Now create another file name *applicationname.py* (or anything depends on what you have in your *app.yaml*. This is the main code to respond to different events in a Wave conversation) in your source directory with the following content:

```
from waveapi import events
from waveapi import robot
from waveapi import appengine_robot_runner

def OnParticipantsChanged(event, wavelet):
    """Invoked when any participants have been added/removed."""
    new_participants = event.participants_added
    for new_participant in new_participants:
        wavelet.reply('\nHi :' + new_participant)

def OnRobotAdded(event, wavelet):
    """Invoked when the robot has been added."""
    wavelet.reply('\nI'm alive!')

if __name__ == '__main__':
    myRobot = robot.Robot('appName',
        image_url='http://appName.appspot.com/icon.png',
        profile_url='http://appName.appspot.com/')
    myRobot.register_handler(events.WaveletParticipantsChanged, OnParticipantsChan
    myRobot.register_handler(events.WaveletSelfAdded, OnRobotAdded)
    appengine_robot_runner.run(myRobot)
```

This is a bit more complicated. The first 3 lines include the Python robot library into your robot:

```
from waveapi import events
from waveapi import robot
from waveapi import appengine_robot_runner
```

The last part of the code is the main function of your robot. It is put at the bottom of the file so that it can access the define functions at the top.

RegisterHandler function register a list of callback function to call when an event happens. A callback function is a function that is passed to another function (in the form of a pointer to the callback function) so that the second function can call it. This is a simple way of making the second function more flexible without the second function needing to know a lot of stuff. By passing different callback functions, you can get different behaviour.

In the example above, we register 2 events with their associated callback functions:

- *WaveletParticipantsChanged*: trigger when list of participants of a Wave conversation changes.
- *WaveletSelfAdded*: trigger when the robot is added into a Wave conversation.

The full list of events can be found here: <http://wave-robot-python-client.googlecode.com/svn/trunk/pydocs/index.html#module-events>

```
myRobot = robot.Robot('appName',
    image_url='http://appName.appspot.com/icon.png',
    profile_url='http://appName.appspot.com/')
```

This is the constructor for the robot. You can specify the name of your robot, profile image, and profile link.

When the matching event occurs, the Wave client will automatically make a call to your application to trigger the event in your code.

The only way your robot can work is via these events. Your team will need to spend some time deciding which event to support and what to do after the event is fired. In CS3216, we expect you to make use of these events to make your robot useful. You shouldn't add more event listeners into your source code just because we ask for it.

Milestone 11: Design your robot to respond to 3-4 events. Explain in detail how these events serve your robot's purpose.

One nice thing about robots is that it can access all the information in a Wave conversation (except for private messages, of course). This makes robot a perfect candidate to do automated tasks. One common task you can start with is to watch the participants' message, find a matching pattern, and respond accordingly - like an emoticon finding robot.

Since Google Wave is an online collaboration tool, sometimes people just throw a lot of stuff into a Wave conversation. Imagine building a gallery in Google Wave with 100 contributors. Everyone contributes 100 images, which makes a total of 10000 pictures. For a human being to collect all these images, it would take a lot of time just to save them to his or her harddisk. This is where Wave robots comes in. A robot can scan through a Wave conversation and extract all images out a lot quicker than a human being. Furthermore a robot can be reused. So if there is an instance where you want to make a second gallery, it would be a lot more easier.

Milestone 12: Extract data from Wave conversation, and explain how you will use the data.

Aside from all the event listeners, you should make use of other APIs to make your robot more useful. Again, you should only use the APIs that are really needed. We will not give credit for poorly-used API calls that do not contribute to a good system design.

Here is a list of existing Wave robot that you can take a look at and find your inspiration:

- <http://wave-samples-gallery.appspot.com/results?api=Robots>

There are also source codes that you can learn from (One of the fastest way to learn is by reading other people's code).

Milestone 13: Make use of 3-5 API functions in your robot. Explain in details how those API functions help to make your robot more useful.

Deploy to Google App Engine

Download Google App Engine SDK for Python here: http://googleappengine.googlecode.com/files/google_appengine_1.3.0.zip

Extract the file to your computer. Make sure you have the correct version of Python on your computer. Fire up terminal/command line on your system and enter this command:

```
appcfg.py update [path to your source folder]
```

You will be asked for your login credential. If there's any error, the script will output accordingly. Otherwise it will say that your application is ready for serving.

There is GUI version of the SDK for Mac and Windows (sorry Linux folks) if you prefer: <http://code.google.com/appengine/downloads.html>

And Java?

We chose to introduce the robots API using Python, because it generally requires less setup than Java, and is easier to explain here.

But, you are still welcome to develop with the robots API using Java. You will need to download the zip files here:

<http://code.google.com/p/wave-robot-java-client/downloads/list>

in addition to installing the Eclipse IDE and Google's plugin for it.

Google has a good step by step tutorial here that we recommend you follow **closely** to get your first Java robot up:

<http://code.google.com/apis/wave/extensions/robots/java-tutorial.html>

The milestones are the same should you decide to use Java. We reiterate here that in terms of grading, we are not concerned about which language you use, so pick the language that maximizes your productivity.

Common for Wave Gadget and Wave Robot

Extension Design Principles

You were asked to consider the Extension Design Principles in designing your extension [0].

Milestone 14: Explain how your extension complies with these design principle. Or explain how and it didn't (as it's not always easy to do so).

Google Wave Extension Installers

An extension installer is a xml file that defines a quick way to:

- Add a gadget to a Wave conversation.
- Add a participant to a Wave conversation (perfect for adding robot, isn't?).
- Tag the current wave conversation with an annotation.
- Create a new wave

In the regular Wave account, follow this tutorial to enable extension installer in your account: <http://code.google.com/apis/wave/extensions/installers/index.html#Production>

The extension installer syntax is fairly simple and we expect you to figure it out by yourself after reading the tutorial here: <http://code.google.com/apis/wave/extensions/installers/index.html>

An extension installer will be the main way to spread your Wave gadget and Wave robot, since most people will not remember your gadget URL or your robot identity. By adding an extension installer, a user can quickly access your gadget/robot within a few clicks and in a graphical way.

Milestone 15: Create an installer for your gadget/robot. It should add your gadget/robot into a Wave conversation automatically.

Embedding API

We have discussed how to build an extension to interact with users in a Wave conversation. Aside from using wave on the Wave website, it's possible to use Wave on your own website. Users with a Wave account can interact with the Wave conversation on your website. Users with the appropriate permissions can view, reply, edit directly to the Wave on your website and it gets updated in real time to another user (be it on googlewave.com or through another site that the same Wave conversation is embedded in).

The embedded wave has the same characteristics as the conversation on the wave client. You can do a few things to it - include gadgets, add a robot to it, and even modify some UI elements to seamlessly integrate with the look and feel of your website. To embed a Wave conversation in your website/blog, you first need to know the Wave id. Each Wave is associated with a globally unique id.

You can use the following method to find a Wave conversation id:

1. Open the Wave conversation in your browser
2. Copy the URL. It will look like this: <https://wave.google.com/wave/#restored:wave:googlewave.com!w%252BWcTtzYqSA>
3. Replace the %252B part with +
4. Your Wave conversation ID is googlewave.com!w+WcTtzYqSA

After you have got the Wave conversation id, open your web page in a text editor and fill this in the `<head>` part.

```
<script src='http://wave-api.appspot.com/public/embed.js'
type='text/javascript'></script>
<script>
function initialize() {
  var wave = new WavePanel('https://wave.google.com/wave/');
  wave.loadWave('YOUR-WAVE-ID');
  wave.init(document.getElementById('placeholder'));
}
</script>
```

The first line includes the wave embed API to your web page, then the *initialize* function will load the Wave conversation into the element with the id *placeholder*. Please note that *placeholder* is an element on your web page, not the wave id.

The *initialize* should only run after the Wave Embed API is fully loaded, otherwise the Wave conversation will not load correctly or behave weirdly.

To make sure that the page is fully loaded, put an *onload* parameter in the body tag:

```
<body onload="initialize();">
```

Now open your web page again, you should see your Wave conversation. By default, it uses standard Wave UI. You can modify some elements to match the theme on your website using the API.

The full API reference can be found here: <http://code.google.com/apis/wave/embed/reference.html>

Google's tutorial on embedding a wave conversation can be found here: <http://code.google.com/apis/wave/embed/guide.html>

Milestone 16:	Make use of Wave Embed API to implement innovative features. Explain to us how Wave Embed API helps you to achieve your project objective. Do note that you need to create an online website (Optional).
----------------------	--

Congratulations! You are now among the first group of people in the world to have successfully developed a Wave extension. Since you are first, we (the Teaching staff) need your help!

Being a new platform, we expect you to suffer a fair share of mishaps and frustrations. :p You should have discovered that developing in a new platform is not all that fun after all. There should be glitches that have yet to be ironed out. Maybe you also have an opinion on how the development experience can be improved.

Milestone 17: What are the shortcomings that you have discovered about the Wave platform? How do you think the Wave platform can be improved?

You might consider publishing your extension to allow people to use it. Google has an example gallery, which you can submit your extension to. To submit your extension to the sample gallery, follow the instruction here: <https://wave-samples-gallery.appspot.com/submit>. Once you have a production-quality extension, can you submit it here: <https://wave-extensions-gallery.appspot.com/submit>.

Detailed Grading Scheme

The grading of the assignment is divided into two components: satisfying the compulsory milestones(70%) and coolness factor (30%).

There are three types of compulsory milestones in this assignment:

1. Milestones common to both paths of development
2. Milestones for the wave gadget section
3. Milestones for the wave robot section

Milestones common to both paths of development

Regardless of the path of development, these milestones are mandatory. This section contributes 25% to the total assignment grade.

There are eight milestones (milestones 0, 1, 2, 3, 14, 15, 16 and 17) in this section. Of these eight milestones, milestone 16 is optional and milestones 2 and 3 are not graded.

Milestones 0, 1, 14, 15 and 17 each contributes 5% to the total assignment grade.

Milestones for the wave gadget section

This section contributes 45% to the total assignment grade.

There are seven milestones (milestone 4-10) in this section.

Milestones 6 and 9 each contributes 7.5% to the total grade. The rest of the milestones each contributes 6% to the total grade.

Milestones for the wave robot section

This section contributes 45% to the total assignment grade.

There are three milestones (milestone 11-13) in this section. Each milestone contributes 15% to the total grade.

If you choose to build both the gadget and the robot, we will be grading both sections and taking the higher grade of the two sections as the final grade. The synergy between the interaction of these two extensions will be taken into account in the coolness factor.

This assignment is due on **27th Feb 2010**. You are expected to deliver the following:

1. Milestone 1: A page or two of write-up on how the Wave extension helps to achieve the objectives of the assignment
2. Milestone 2: A page or two of description on your target users and your marketing strategies for your application
3. Completion of all remaining compulsory milestones and write-up explaining how we can see that the milestones were achieved.
4. A short write-up pitching your application to the teaching staff, i.e. convince us that your application is so good that it deserves all 30% of the coolness factor points. Restriction: no longer than 2 pages (A4), 12pt. font, Georgia or equivalent.

Mode of Submission

The following is the list of deliverables:

1. Source code: compress all the source files into a single archive (zip/rar/tarball), maintaining the directory structure of the source files.
2. Provide us the URL to your installer, i.e. your gadget/robot must be accessible online somewhere. If you built a gadget, you're required to provide the URL of the gadget (just in case your installer doesn't work). Similarly, if you build a robot, you're required to provide the robot identity. If you do both, submit both and write a short description on the sequence of adding gadget/robot (e.g. your app only works correctly if the gadget/robot is added first).
3. If your gadget/robot only works in sandbox environment, please state it clearly or we might not be able to run it. By default we test your gadget/robot in regular Wave environment.
4. A short readme file containing the list of group members, including matriculation numbers, name and a description of the contributions of each member to the assignment. Make sure that your application name is clearly written in the README file. The README file should also contain all the necessary write-ups for the milestones for this assignment.

Archive all deliverables into a single archive (zip/rar/tarball) and name it assignment2-[GroupNo].zip/rar/tarball.

As a reminder, if you choose to build a wave gadget, you need to complete seven mandatory milestones. If you choose to build a robot, you are expected to complete three compulsory milestones. In addition, there are six milestones (excluding milestone 15) common to both paths of development.

Good luck and have fun!

References

- [1] Google. The complete guide to google wave. http://completewaveguide.com/guide/Meet_Google_Wave.
- [2] Google. Wave extension design principles
. <http://code.google.com/apis/wave/extensions/designprinciples.html>.
- [3] G. W. A. Team. Connect four
. http://wave-samples-gallery.appspot.com/about_app?app_id=36022.
- [4] G. W. A. Team. Yellow highlighter
. http://wave-samples-gallery.appspot.com/about_app?app_id=35019.
- [5] Wikipedia. Google wave. http://en.wikipedia.org/wiki/Google_Wave.