

# Introduction to the

# Durandal Framework v19.0

Revision 3304 – Logan Stromberg – 5 June 2019

**Durandal** is a framework and an API for developing natural language dialog systems that are both portable and extensible. It is currently being used as a “Cortana Emulator” for prototyping new scenarios and investigating potential design changes for the Bing Conversational Understanding system. Apart from that, it is also a fun project which any developer can use to add natural dialog to any application

# Overview of the system

The Durandal system is divided into three main parts: Language Understanding Service, Dialog Service, and Client. Each one is a separate program with a distinct role.

**Language Understanding (LU)**: Performs the work of discovering out *what a user said and what they meant by it.* This component will dynamically load language training data and use it to generate models for intent classification and tagging. It provides annotation services accessible through HTTP. As an example, it could accept the string “remind me to take out the trash at 6:00” and parse it into:

Conversation Domain = reminder

User’s Intent = create\_reminder

Reminder title = “take out the trash”

Reminder time = “at 6:00”

Timex value: 2018-05-11T18:00

**Dialog Service (DE):** Performs the work of *executing some action and generating a valid response to what the user said.* The service itself is a framework which handles the complicated dialog state logic, and a set of plugins that can be triggered and execute any arbitrary behavior in response to the user’s utterance. This is the area which a developer will be most concerned. Adding new behavior to the system is a matter of creating language models (in LU) that will trigger certain actions, and then writing the plugin code (a standalone DLL) that will respond to those actions, by consuming the structured classifier output which LU produces.

**Client:** Provides a method of capturing user’s input (text or speech) and presenting the results to the user. Usually this is comprised of a text input box, a microphone, and an area for displaying HTML. For some prototypes this will simply be a Web browser. The dialog system attempts to be as client-agnostic as possible, which means that it should ideally be able to provide a useful response to text-only clients, audio-visual clients, zero-display clients, etc.

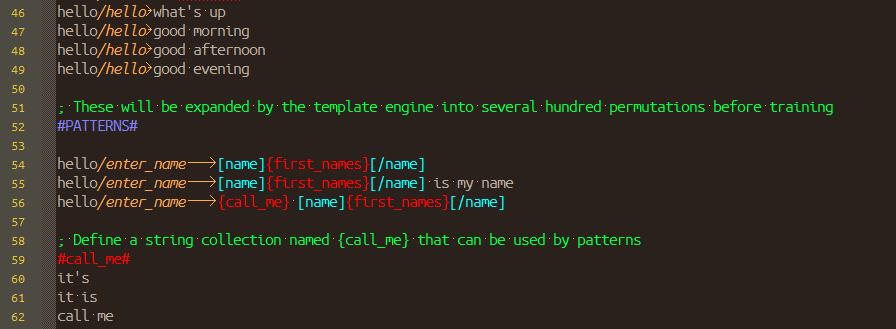
# Hello World

## Building a basic plugin

Now that the overview is finished, let’s open up the project and see how to start writing your own plugins. The DurandalSDKSample solution contains the code for 2 plugins, the “Hello” example, and a simple Bing Search plugin.

### Step 1: Create some training data

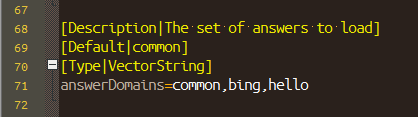
Training data for LU comes from a series of template files found in /bin/training. These template files use a custom syntax which is self-documented in hello.template. The exact name of the file is irrelevant; all .template files in the training directory will be processed. There is another folder for validation data – this folder is optional and contains template files in the exact same format as the training data.



Training data template

### Step 2: Add your domain to LU and Dialog configuration

The main configuration files for the program are found in /bin/DialogEngine\_config.ini and /bin/LanguageUnderstanding\_config.ini. For the most part you should not need to modify these values, except for when adding new plugins. In order to tell the system which plugins (and conversation domains) to enable, modify the answerDomains= line in LU and pluginIdsToLoad= in Dialog configuration files, adding your new domain/plugin ID to the list.

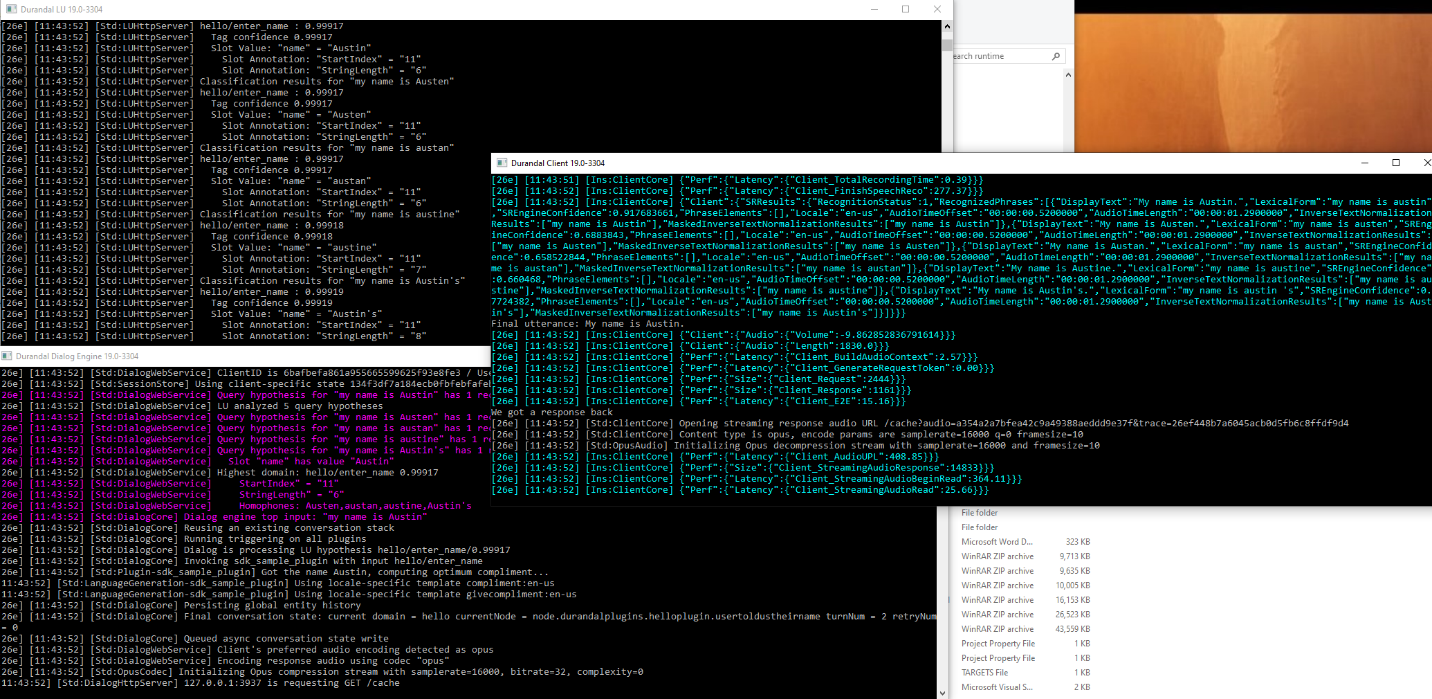


### Step 3: Write a plugin, and compile it

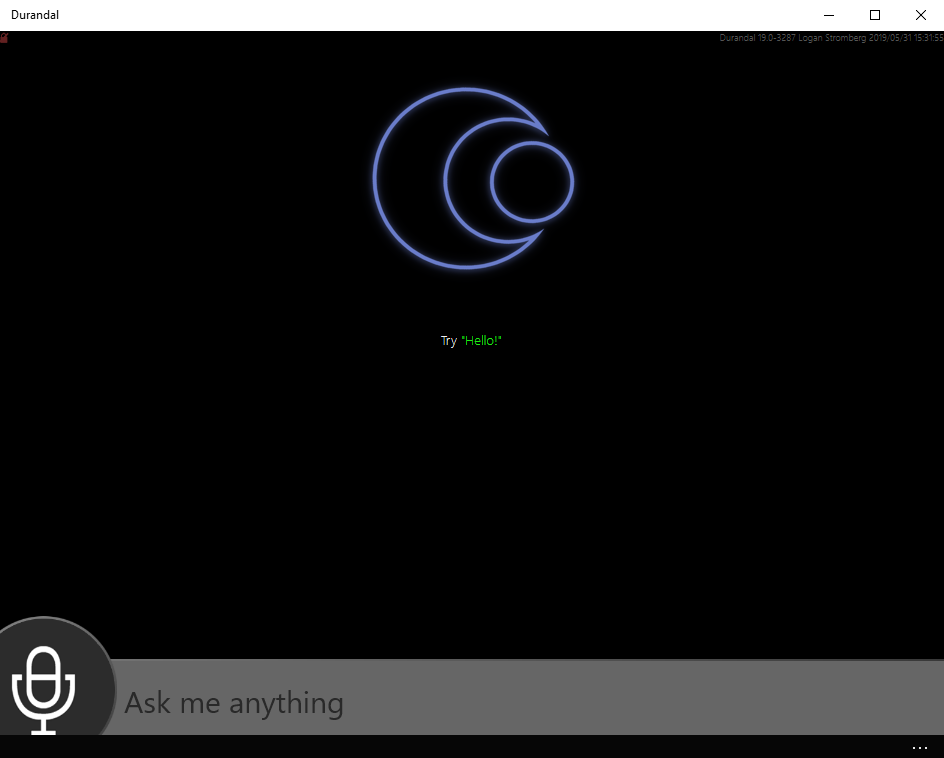
Plugins must inherit from DurandalPlugin (found inside Durandal.dll) and need to override a few functions in order to function. At the minimum, they must use a constructor which specifies their unique plugin ID and the LU domain they handle (“bing”, “places”, “alarm”, etc.) and an implementation of the Execute() function. A sample plugin is found in /DurandalPlugins/HelloPlugin.cs, as well as self-documentation on how to use several of the basic functions of the API.

### Step 4: Run the code

To see the plugin in action, start up each component by running first /runtime/LanguageUnderstanding.exe, then /runtime/DialogEngineConsole.exe, and finally /runtime/ConsoleClient.exe. Enter some text or speak to the client, and watch your query run through each component and return your results. The example plugin is configured to listen to “Hello”. You can also view it using a plain Web browser by navigating to <http://localhost:62292>



The code in action



And running on the UWP client

As a side note, LU and DE have their own debug consoles built-in. Simply type a query into their respective windows to view the results.

Try starting a conversation with “Hello” and see what happens.

# Todo: fill in more documentation!

# Language Understanding

## Template language format

## Annotators

## Model configuration

## Crosstraining

# Dialog

## Plugin inputs

## Conversation trees

## Plugin responses

# Language Generation

## Template Engine

## Statistical Engine

# Clients

## Client adaptation

## Writing your own client

## Client-side actions

## Client-side UI

## Authentication and scope

# Advanced Usage Patterns

## Script-driven dialogs

## Graceful fallback

## Lexicographic search and indexing

## Botlet-style invocation, callbacks and continuations

## Full conversation control

## Custom audio pipelines

## Explicit continuations

## Keyword spotter manipulation

## Sentiment analysis

# Reference

## Core class and function documentation

## Glossary

**Answer** – A dialog plugin which provides responses to a particular set of utterances. Common answers include Bing Search, Reminders, Weather, and Time.

**Domain** – A logical grouping of similar conversation topics. For example, the “web\_search” domain might include things such as image search, local search, video search, etc., all handled by one answer (Dialog engine plugins are mapped to domains 1:1)

**Intent** – A description of the action the user might intend with a particular utterance. The phrase “where is the nearest pizza place” might have the intent “find\_restaurant”, and “where am I?” might have the intent “show\_map”. All classified utterances will have a domain and intent, which is the minimum to specify “what answer should handle this?” and “how should it react?”

**Language Model** – Here, it is a statistical model which is trained using natural language phrases and allows the system to predict and classify the user’s intentions. The language model in Durandal consists of a Max Entropy binary classifier for domain/intent, and a Conditional Random Field (CRF) tagger to extract slot values.

**Multiturn** – A conversation for which context is preserved between queries, guiding the user to allow completion of some task, or allowing elaboration and refinement of the original query.

**Slot** – A string value that has been tagged in a user utterance. For example, the phrase “My name is Hisami” might have a slot called “name” with a value of “Hisami”. More advanced slots can represent time expressions, ordinals, hypotheses, or structured data such as Xml to pass around data within the dialog system

**Turn** – A single request-response pair within a conversation. All conversations are modeled as a series of turns, meaning that the user is speaking directly to the computer and the computer provides exactly 1 response for every user utterance.