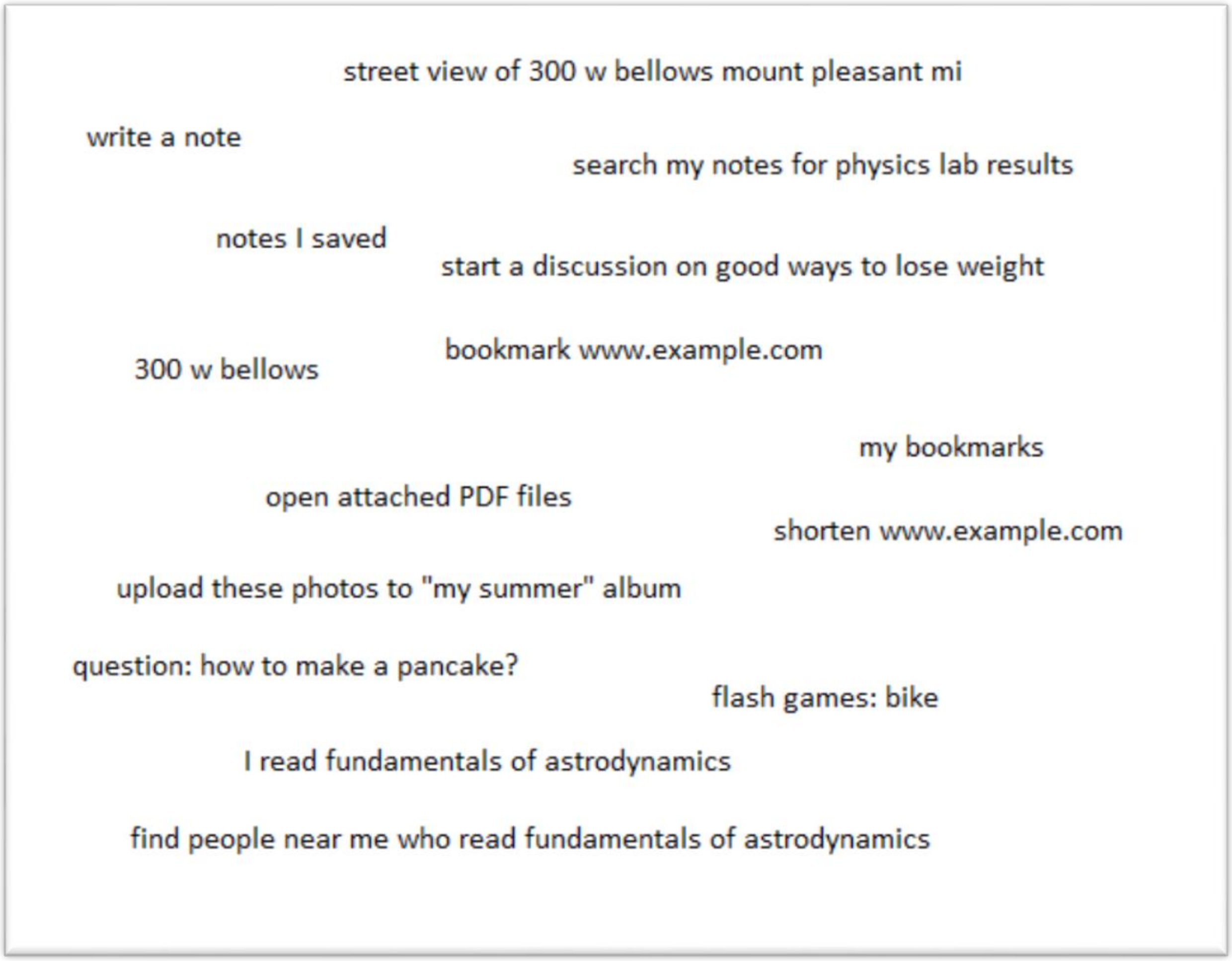
Nerddy

DEVELOPER MANUAL

Contents

What is Nerddy?

Nerddy is a platform that offers end-users a new and easy way to access information and perform operations on the Internet through natural language. It combines a powerful command-line interface and a graphical user interface on one page. Nerddy connects users and web applications on one page and provides APIs to these web applications to allow them to handle queries and deliver output to users. Nerddy uses a sophisticated query interpretation system and allows for dynamic, robust queries. Users are not required to install apps because Nerddy can recognize what apps can handle a query based on instructions registered on the system by the apps. Anyone can create a public app on Nerddy. Apps are instructions that tell Nerddy how to recognize and handle queries. In return, Nerddy dedicates a significant portion of revenue to applications through the rev share program in addition to allowing and facilitating different forms of monetization. Below are sample queries that a user might make.



street view of 300 w bellows mount pleasant mi

write a note

search my notes for physics lab results

notes I saved

start a discussion on good ways to lose weight

bookmark www.example.com

300 w bellows

my bookmarks

open attached PDF files

shorten www.example.com

upload these photos to "my summer" album

question: how to make a pancake?

flash games: bike

I read fundamentals of astrodynamics

find people near me who read fundamentals of astrodynamics

Please refer to the end-user manual for more details on what Nerddy is.

Flow of Events

The figure below describes the flow of events on Nerddy beginning from when an end-user submits a query.

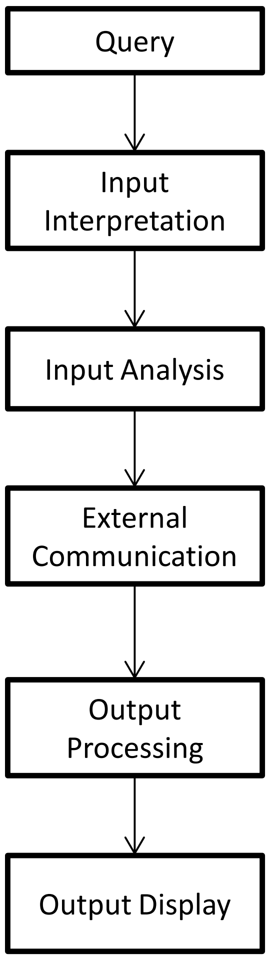


Figure 1: Flow of Events

The purpose of each stage is described in the table below.

Query	Queries are the text input entered by the user to complete an action or get information. Queries may contain attached files.
Input Interpretation	The stage where Nerddy looks for recognized applications on the platform that could handle the query. Input interpretation relies on five different types of keywords, query length, entity recognition, and attached files.
Input Analysis	The stage where data and entities are extracted from a query referring to critical word based instructions. Critical words are a special type of keywords declared by

	apps. These keywords are likely to exist in proximity of input values of interest in a query. A familiar example of a critical word is a unit of measurement such as kg or lb. Referring to instructions registered by the app, Nerddy will extract values of interest in a query and store them to variables that can be accessed in later stages.
Input Processing	The stage where programmatic manipulation of extracted data and queries is made possible. This is also the stage where the app instructions must specify what data should be made accessible in the next stages.
External Communication	The stage where the query, attached files, extracted variables, new variables, and necessary user data are transferred to the end point/third party server associated with the app. Applications may choose not to return a response immediately and use the Notifications API (Output) and Wallet API to make asynchronous requests to send output or payment requests to the user.
Output Processing	This is the stage where all data made accessible in the input processing stage and the response returned by the third party application during the external communication stage, if any, could be programmatically processed in order to display the data on the user's front-end. This is a highly optional stage for apps that use the external communication system and it's recommended that you use the Notification API directly.
Output Display	This is the stage where output is displayed to the user. Output will be cached on Nerddy's servers to allow users to save sessions and access saved and recently lost sessions. You may disable session retrieval for your results as discussed in the Notification API manual.

APIs that Apps Can Access

Notifications/Messages API	Allows apps to send output, content, and normal notifications to users. The system also allows apps to reach new users through sidebar notifications.
Wallet API**	Allows apps to send payment requests to users. The system supports automatic (recurring) payments and shipping options.
Profiles API*	Allows apps to create app-specific profiles for users which can be viewed by other users utilizing the notifications system.
User Search API**	Allows apps to search for users using specified criteria.
User Data API**	Allows apps to access user data asynchronously.

Connections API*	Allows apps to create connections between users to allow for entity recognition.
PINs API	Allows apps to store small values that could be accessed by Nerddy's front-end output display system. Typically pins are used for user authentication such as when a page that uses the GET method is iframed in the output.

*Requires granted permission to use the API.

**Requires granted permission to use the API and permission for each input parameter.

Building a Nerddy App

You don't need a special account to build an app on Nerddy. Anyone can use dev commands (which are discussed later in this manual) to submit an app to the platform.

Types of apps on Nerddy

• System apps: These are apps that are built into the platform such as the sign out app, web search, search history, session retrieval app, etc.

• Dev apps: These are Nerddy-owned public apps that can be used by developers to submit, retrieve and edit app files, access traffic stats and earnings, etc.

• Public apps: These are apps created and submitted by users and can be used by everyone. They consist of two components:

Parent app: Represents a collection of child apps. At least one child app must be created and defined under a parent app. Put simply, a parent app is access information shared by child apps under it. The parent app ID must be registered before child apps.

Child apps: These are app instructions created and submitted by the parent app owner and registered under parent apps. They don't have a special global id or access key because they use that of the parent app.

App Creation Checklist

1) Develop a web application on your server.

2) Read all the documentation manuals and create end-points for each Nerddy API you plan to use.

You may use a public API offered by another party either directly on Nerddy or on your server. However, your app will not be considered an official app for the third party content provider but can still gain approval.

2) Purchase a parent app ID using dev commands and create a child app under the parent app.

3) Create a .nrd file for the child app. The .nrd file should contain app instructions as you will learn later.

4) Submit the .nrd file. Review process could take anywhere from 48 hours to 5 days. Quite often, app instructions are modified by reviewers for optimal performance.

5) You may request additional special permissions through the support system after an app is approved. There is no guarantee that the requested permissions will be granted.

6) Create content for your app on Explore Apps page.

7) Advertise, maintain, and monetize your app.

Dev Commands

The table below lists dev commands that every developer should be aware of. You can enter these commands on Nerddy just like you search for content.

dev: purchase a parent app id	Returns a payment request with a physical address being required. After payment is complete, a parent (globalid) app ID will be generated.
dev: create child app id under [globalID] *For example:* dev: create child app id under uf3lfgjf7fgvm	Registers a child ID (innerid) under a parent app. Any app needs at least one innerid and one globalid.
dev: my user id	Returns your userid.
dev: my apps	Returns list of parent and child app IDs registered and the status of the associated app submissions.
dev: view child apps under [globalID]	Returns list of innerid(s) under a globalid.
dev: submit new app [globalID] [innerID] Note: A .nrd file containing app instructions must be attached. Please note that globalID and innerID must be delimited by a space.	Submits the app for review. Please make sure that all required app instructions are included in the .nrd file. Not all instructions are required.
dev: submit edit [globalID] [innerID] [identity_verification_code] Note: A .nrd file containing updated app instructions must be attached.	Submits a new version of an existing approved app for review. Please make sure to use the same identity_verification_code you specified in the original .nrd file for that app.
dev: download latest app file [globalID] [innerID]	Exports the last approved and reviewer modified .nrd file. Submitted .nrd files may be edited in the review process. It's important that you build on the reviewed version for future edits.
dev: get app auth info for [globalID] [innerID] [identity_verification_code]	Returns parent appid, accesskey, and vericode. This is important for API access.

dev: api access for [globalID]	Returns API access privilege for the app.
dev: view app stats for [globalID] from [date] to [date] *For example:* dev: view app stats for sjh74akms2w from Jan 15 till January 31 The stats dev app was built to be flexible with natural language. You may word the dev query as you wish. Another common way to word it: dev: view app stats for [globalID] for [fixedperiod] Note: [fixedperiod] may be specified as today, yesterday, or this month	Returns a zip archive that contains excel files for each child app under the parent app and an excel file for the parent app (AKA sum file). The excel files contain date and time of access, user IP address, userid, and sign up country. Note that parent app stats only count unique users for each day. If a user uses the app multiple times on the same day or uses multiple child apps of the same parent app, only the first will be shown on the parent app stats (the sum file). On the child app stats; however, every request will be shown.
dev: view my wallet balance	Returns the wallet balance for each parent app that you own. Wallet balance doesn't include advertising rev share and only includes unpaid earnings made from sales through the Nerddy Wallet system.
dev: payment history	Allows you to retrieve a list of payments made to you.
dev: set payment method	Allows you to specify payment method to receive payout.
dev: set wallet action url for [globalID] as https://www.example.com/abc.php	Allows you to set the endpoint where Nerddy sends confirmations after successful payments from users.
dev: set wallet subscription notify url for [globalID] as https://www.example.com	This command will define the page where Nerddy sends notifications for subscription start, termination intent, and termination.
dev: set messages endpoint for [globalID] as https://www.example.com/abc.php	Allows you to set the endpoint where Nerddy sends messages from users. Refer to the Messages API section in the Notifications/Messages API manual.
dev: publish new article on app explore for [globalID][innerID] Note: A .txt file must be attached that contains an HTML body of the content to be published. It may not contain JavaScript or PHP. On the first line, please specify article	Publishes new content on the app's blog page on Nerddy Explore. Submissions will only be published after human review. Article order specifies the position of the article relative to other articles from the top.

order and article title in this format: <articleorder>1</articleorder> <articletitle>This is a Title</articletitle> <htmlcontent>HTML Body</htmlcontent>	Please carefully proofread and review before submission. If you want to delete or modify an existing article, you will need to contact the developer support.
dev: list dev commands	Returns this list of dev commands.
dev: documentation	Returns app development documentation.
dev: support	Returns contact information for developer support. Please contact the appropriate department to avoid unnecessary delays.

These dev commands will be useable once the rev share program is open:

dev: view point value for [globalID] [month]	Will return the value of each request made to your app. Value of each request is determined based on total advertising revenue and algorithmic estimates of how much your app contributed to the growth of the platform over the period.
dev: view rev share earnings for [globalID] from [date] to [date] dev: view rev share earnings for [globalID] for [fixedperiod]	Will return the rev share earnings for the app for the specified period.

Components of a .nrd App File

Please download a sample .nrd app file from this page. We highly recommend that you view the .nrd file side-by-side with this documentation using a text editor. Data format is JSON. Sections marked by * are required and must be present in the .nrd file. Other sections are optional.

appid*

The appid section presents app identification. Make sure to register a parent appid and a child appid first. The table below lists the elements of the appid section.

globalid	Registered globalid (parent app ID).
innerid	A child ID registered under the globalid.
userid	Your userid (NOT your username or email). Use the user id dev command to find it.
type	Can be “new submission” or “update”

owner*

The owner section serves to verify your identity when you submit updated versions of the same app or when you contact the developer support.

email	Administrator e-mail.
identity_verification_code	A passphrase that is required to access app auth info and make future changes to the app.
contact	Contact email or phone number that will be given to end users who reach out for support for issues related to your app.
parent	If you are a company, specify company's business name. If not, specify "NULL"

about*

The about section describes your application.

title	The title of your application. Visible to users in attribution and Nerddy Explore.
description	Describe briefly and accurately what your app does.
category	Describe your app category in two words or less. Example categories: travel, messaging, calculator, search, live video, photos, etc.
language	State the language that your application can interpret. Currently we only support English.
type	Specify as "public"
minimum_age	Specify the minimum age that your app is targeting. Users below the minimum age will still be able to use the app. However, the app won't be advertised to them.
user_location_requirement	This feature is not available yet. Set to "All"
logo	Logo of your application. Recommended width is 180px. Visible to users in attribution.
small_icon	A 16 x 16 icon. Will be shown as favicon when user is asked to select an app if more than one app can handle the query.
show_attribution	Specify "yes" to show an attribution box or "no" for otherwise. An attribution box typically appears on the top right of the page telling the user what app handled the query.
attribution_links	Specify the anchor texts and URLs in the format shown on the sample .nrd file.
showonexplore	Can be 1 or 0. If 1 is specified, the app will have a blog on Nerddy Explore and will be advertised to users. If 0 is specified, the app will be hidden from Nerddy Explore's index.
official	Specify "yes" if you are the owner of the third party APIs associated with the app. Specify "no" if

	you are using public APIs to get the output.

permissions*

The permissions section allows you to request access to APIs that require system granted permissions to access. Please note that you should only request access to APIs and parameters that you need. Your whole app may be rejected if you request access to parameters you obviously don't need for your app to function.

wallet	Request wallet permissions in this form: "wallet": {"api":"allow","limit":"50", "recurring":"allow"} If your app needs access to the wallet API, specify "allow" to the api parameter. The api parameter can be set to either "allow" or "disallow" Limit specifies the maximum amount your app can request per transaction. If your app is subscription based, you will need to request access to recurring payments in the wallet system.
user_data_api	Request User Data API permissions in this form: "user_data_api": {"api": "allow", "parameters":"userprofession, usercity, userzip"} Refer to the User Data API manual for a list of supported parameters. You must specify each parameter you want access to.
user_search_api	Request User Search API permissions in this form: "user_search_api": {"api" : "allow", "limit":"8", "criteria": "usersearchhistory, userschool, userstate"} Refer to the User Search API manual for a list of supported criteria. Your app needs to obtain permission to specific search criteria before you can use the User Search API to search for users using the criteria. The limit parameter specifies the maximum number of users you can obtain in a search result.
connections_api	Request access to the Connections API in this form: "connections_api": "allow" The connections_api parameter can be set to either allow or disallow.

profiles_api	Request access to the Profiles API in this form: "profiles_api": "allow" The profiles_api parameters can be set to either allow or disallow.
messages_api	Request access to the Messages API in this form: "messages_api": "allow" The messages_api parameters can be set to either allow or disallow.

sessionlock

The session lock section specifies the endpoint and authentication information that should be sent along with the query when the user is locked in a session with your app. A locked session or a continuous session means that queries that the user enters once the session is created will be sent to your application's specified endpoint instead of being interpreted. A session is terminated when the user enters "end" query. Your app must clearly state to the user in every response that they are in a session with the app and to quit the session, they should enter "end" as a query. Failure to do so will result in app suspension and possibly termination.

session_api	Specify the end-point where the query, userid and authentication information will be sent
session_auth	Must be in JSON format as in the sample .nrd file given. Whatever you specify here will be returned under auth whenever a query is sent to your end-point.

The session system will make a request to the end-point with this body format when a query is made by a user who is in a session with your app:

```
{"auth":{"whateveryouspecified":"whatever", "whatever":"whatever"},

"body":{"sessionid":"xxxx","userid":"xxxx","userquery":"query entered by the user",
"File":"http://www.nrdcdn.com/example.zip, http://www.nrdcdn.com/example2.zip",
"FileName":"my_music, cat"}}
```

If the query contains attached files, the system will send direct download URLs to each attached file in zip format with a random filename. If multiple files are uploaded, the URLs will be comma delimited. Original filenames will be given under FileName (also comma delimited). Sessionid, userid, and userquery will always be provided. Sessionid is different from usersessionid. Usersessionid is different for each tab. Sessionid is not as it represents a session with your app.

interpretation*

Interpretation provides criteria required to be met by queries that should be passed to the app. The table below lists the elements of an input interpretation record. You may create multiple records under interpretation. The first record should be "1":{}, second should be "2":{}, etc.

main_substring	A substring that must always exist in a query. Can be more than one word. May be left empty.
word_count_condition	Can be "less" or "greater" or "equal"
word_count	Specify the word count that the query must meet based on the word count condition. If query can be of any length use word_count of 0 and specify word_count_condition as greater.
required_keywords	Specify sets of keywords that may exist in a query. A set represents a keyword and its synonyms. You may specify more than one set as in the sample .nrd file. If one keyword in a set exists in the query, the set will be satisfied. A keyword may be made of more than a single word and may contain characters like colon. Keywords can be substrings. Substring keywords must be surrounded by square brackets as in [meter] where meter is the substring keyword.
rkp	Ratio of sets that MUST be satisfied in a query to the total number of sets in the required_keywords multiplied by a hundred.
may_exist_keywords	Specify keywords that may exist in a query. These keywords don't count in the denominator of the rkp equation. However, if one exists, mkp will be added to the calculated rkp.
mkp	Specify the value of mkp that should be added to the rkp if a may_exist_keyword is present in the query.
required_mentions	Specify a single type of entity that must exist in the query. Note that not all entity types that are supported in input analysis are supported in required_mentions. Required_mentions serves as an additional interpretation layer and can be set to:

	• *none* The query doesn’t have to contain any special entity type. • *number* The query must contain a number. • *connection(specifyrelation) or connection(ALL)* The query must contain the name of a person that the user might know. Refer to the Connections API manual. Note that connection will also recognize the user connections mentioned in the query and will store them under #!Interpret_Connections!# variable. Different users will be separated by commas. If two connections have the same name as one mentioned in the query, Nerddy will return both between parentheses. • *username* The username of another Nerddy user. The userid(s) associated with usernames detected in the query will be stored under #!usermentionids!# variable while the original usernames will be stored under #!usermentions!# separated by commas. • *useremail* The email associated with the account of another Nerddy user. The userid(s) associated with useremails detected in the query will be stored under #!usermentionids!# variable while the emails will be stored under #!usermentions!# separated by commas. • *url* The query must contain a URL. • *phone* The query must contain a phone number.

restricted_words	Specify words that must not exist in the query. Words must be comma delimited. For substrings, surround the substring with square brackets.
create_session	Can be “yes” or “no”. If yes is specified, the user will be put in a continuous session with your app. Queries entered by the user will be sent to the session_api endpoint specified under sessionlock.
session_length	Limit the length of the session to a number of queries. Must be an integer.
file	Can be “yes” or “no” or “maybe”. If yes is specified, this means that your app is always expecting a file to be attached to the query. If no is specified, the query may not contain attached files. If maybe is specified, the query may or may not contain attached files.
file_type	Limit the attached file support to certain extensions. Must be comma delimited. Don’t put a period before the extension.
disallowed_file_types	Disallow certain file types that your app doesn’t process. Similar to file_type, it must be comma delimited.

analysis

The table below lists the elements of a critical word based input analysis record. Similar to interpretation, one app may have multiple records. Records are order sensitive. If more than one record uses the same variable name, the new record associated value will overwrite the older one. If your app uses multiple critical words, you will need to create multiple records. Each critical word should have a unique cwid.

cwid	A 6 or more digit unique integer that identifies the critical word. This is necessary to be able to make changes to the record.
cw_type	Can only be 1 or 2. Use 1 if the critical word is one word and has a fixed position in the query. Use 2 if the critical word can appear anywhere in the query or if the critical word is made of multiple words. In other words, cw_type = 1 means that the critical word is a position sensitive word.
cw_position	Always specify 0 if you set cw_type to 2. If cw_type is 1, cw_position is the required order of the critical word in the query. Specify 0 if the

	critical word will always be the last word in the query. Specify the order otherwise as an integer. For instance, if the specified critical word should always be the second word in the query, specify 2. If the critical word doesn't exist in that position and exists elsewhere in the query, it will not be recognized. Observe the queries below. Suppose you are building an app to help users download movies you have the rights to share. There are many ways the query that your users make could be worded. You want them all to be analyzed by your app. 1) download fingerman the movie 2) fingerman the movie download 3) I want to download fingerman the movie In the examples above, the developer should account for the various ways the input could be put. Assume "download" is the critical word and "fingerman the movie" is the input we are trying to extract. We don't know how the user will word the query. Notice that the input location is not always fixed relative to the critical word. The way this analysis could be handled on Nerddy is: • Create three input analysis records for download. The first record should be position insensitive. The second and third records should be position sensitive. Specify cw_position = 1 for the second record and cw_position = 0 for the third record or the vice versa. • The input analysis system is record order sensitive if you specify the same variable_name and the same critical_word for different records. The order in which you specify the records is important. In the example above, the position-sensitive critical word input will override the position-insensitive one allowing your app to get the correct input. If the useful input will always appear in a fixed position relative to a critical word that could appear anywhere in the query, you don't need to worry about this. Set cw_position to 0 and cw_type to 2.
variable_name	Specify the name of the parameter that will hold

	the extracted input data associated with the defined critical word.
critical_word	Specify the critical word. Can be a one word as in “download” or a combination as in “rent in” and “view:”
alternative	Specify synonyms to the critical word. Must be comma delimited.
must_exist	Can be 0 or 1. If set to 0, interpretation will not fail if the critical word doesn’t exist in the query. If set to 1, interpretation will fail if the critical word doesn’t exist in the query.
mention_number	If the critical word could appear multiple times in the query, which one should be the critical word? If you are not sure, set this to 1 (first mention) -1: strictly last mention – i.e. the cw must appear multiple times in the query and it should always be the last one. 0: last mention – i.e. the cw may appear multiple times in the query. If it does, the one farthest from the first word will be the critical word. 1: first mention – i.e. the cw may appear multiple times in the query. If it does, the first mention of the critical word in the query is the critical word. 2: second mention – i.e the critical word must appear at least twice in the query and the critical word is always the second mention. n: nth mention.
input_type	Supported Input Types: "word" : Only one word "more than one word" : One or more words from critical word to second_limit "number" : Integer or float. Nerddy also recognizes written numbers from one to ten and converts them to INT when stored in a variable that carries a number. "array of numbers" : Comma-delimited numbers. For example, convert 18.2, 121, 189 cm to m. "location" : If input type is location, Nerddy will recognize addresses and cities and store them in the defined variable in this format: Street: 300, Route: West Bellows Street, Town: Mount

	Pleasant, City: Isabella County, State: Michigan, Country: United States, Postal Code: 48858 "date" : If input type is date, Nerddy will recognize dates and return them in this format: Day: 12 Month: 2 Year: 2017 "url": Recognizes one or more URLs. The URLs will be stored in comma-delimited format. "phone": Recognizes North American phone numbers put in any format such as 0123456789 or 012 345 6789. "email": Recognizes one or more email addresses. The emails will be returned in comma-delimited format. "connection (relation) ": Recognizes entities of type [relation] who were connected to the user through the Connections API by the app or by other apps (public connections only). The userids of the entities will be stored under the associated variable. If more than one connection has the same name mentioned in the query, Nerddy will return the userid of all of them between parentheses. You should then ask the user which user exactly they meant. "username": Recognizes usernames. The userids of the recognized usernames will be stored under the associated variable (comma-separated). “useremail”: Recognizes e-mails of registered users. The userids of the recognized usernames will be stored under the associated variable (comma-separated).
input_location	Specify the position of the useful input you are trying to extract with respect to the critical word. This parameter can be set to either "before" or "after" Before: the input will always exist right before the critical word. After: the input will always exist right after the critical word. Exception: For input_type = number and input_type = url, input_location specifies the

	location relative to the critical word but doesn’t mean the input will be right after or before the critical word.
input_limit_type	Input limits are useful for "more than one word" input type. It can be set to either "none" or "word" or "count" If set to none, this means that the input is expected to be unbounded in the query. If set to word, this means that the input can exist only from the critical word up to a defined word that should always exist in the query referred to as second_limit. If set to count, this means that the system will read count words after or before the critical word as the input.
limit_count	Set to 0 if input_limit_type is anything other than count. Set to the number of words you wish to take as input if input_limit_type is count.
second_limit	Set to 0 if input_limit_type is anything other than word. If input_limit_type is word, set second_limit to the word you use as the second limit. You may specify synonyms to the word. Synonyms must be comma delimited. The second_limit word will not be part of the input.
second_limit_mention	Can be 0 or 1 or 2. Set to 0 if input_limit_type is not word. If input_limit_type is word, specify which mention of the second_limit should be considered the second_limit if the second_limit word exist twice in the query. If set to 1, the first mention of the word in the query will be the second_limit.
ignore_word_list	Specify a list of comma-delimited words that may exist after the critical word that should be removed from the beginning of the input.
no_mention	Specify a list of comma-delimited words that must not exist in the query. If one of the specified words exists in the query, the system will not do analysis for the critical word and will proceed to the next critical word.
connection_distance	Valid for input_type = connection (relation) and input_type = number. Set as 0 for otherwise. The connection_distance is an integer that limits how far from the critical word the system should look for numerical input or human entities.

input-processing*

The table below lists the components and allowed statements in input processing.

variable_declaration{}	Use variable declaration to define new variables that were not defined in the input analysis stage and are not system reserved. Below is a list of possible variable declaration statements. Variable declaration statements don't end with a semicolon. Variable declaration statements must be inside variable_declaration{}

#!variablename!#	Simply mention the variables. Each variable should be listed on a new line.
#!variablename!# = value	You may assign values to variables. No semicolon needed. Values can be numbers or sentences

query_operations{}	Use query operations to search the query for the existence of certain words or substrings. The system supports AND and OR operators. Variable assignments under query_operations{} must end with semicolon. Example statements: If [pet] OR [dog] OR "cat" exists then{#!pets!#=1;} If "one night" AND "free" exist then [#!specialoffer!#=1;} If "two nights" AND "free" exist then {#!specialoffer!#=2;} [] specifies a substring "" specifies whole words
variable_operations{}	Variable_operations carries out operations based on input analysis variables and newly defined variables rather than the whole query. Example statements: if #!operation!#==1 then{#!measure_in_m[]!#=#!length[]!#/100; #!measure_in_mm[]!#=#!length[]!#*10; #!measure_in_cm[]!#=#!length[]!#;} In the above example, the input type is array of numbers. Arrays must contain empty square brackets before the last exclamation mark in the variable name. The example above is from a unit conversion app where length is given in cm. For example: convert 19, 132.12, 181 cm to m More Examples:

	If #!weekstodays!#==1 then{ #!days!#=#!weeks!#*7;}
	#!newwidth!# = 2*sqrt(#!width!# * 5);
	#!viz!#=<a href="#!video!#">Click here</a>;
	
	A more complex example:
	if #!p!# <= #!w!#*#!cat!# or #!t!# >= 1 then{
	    if #!x!# != 0 then{
	        #!res!# = thickness is #!t!# centimeter;
	        #!z!#=#!x!#/1000;
	    }
	    #!y!# = #!w!#*#!l!#;
	    if "m" == units(#!t!#) then{#!y!# = #!y!# meters;}
	}
	
	units (#!variablename!#) returns the word after the value associated with the variable in the query. Only one level of nesting is supported. For loops are not supported in the input processing stage. You can handle more advanced processing on your own server. The input processing stage only provides support to basic statements that are common in query NLP.

return{}	This is the only part of input processing that is not optional and is required for all apps. You must list the variables you are interested in transferring to the next stages under return{} delimited by a semicolon. Example: return{ #!orderquantity!#; #!apples!#; #!userquery!#; #!userqueryid!#; #!usersessionid!#; #!userid!#; #!userpin!#; #!usersubid1!#; #!usersubid2!#; #!userfullname!# #!File[]!#; #!FileName[]!#; if #!operation!#==1 then{#!mpa!#;} if #!operation!#==2 then{#!ksi!#; #!system!#;}} Please refer to the User Data API for a full list of supported user variables that are associated with account data. #!userid!#, #!userquery!#, and all variables that start with “user” are system reserved. Important system reserved variables:

	#!userid!# : returns the user ID. **#!userquery!#**: returns the query made by the user. **#!userqueryid!#** : returns the value associated with the ID of the query made by the user. **#!usersessionid!#** : returns the value associated with the ID of the session where the query was made. **#!userpin!#**: returns the user pin stored by your app for the user using the PINs API. **#!usersubid1!#**: returns the value for under the usersubid1 stored for the user by your app through the PINs API. **#!File[]!#** : returns URLs to access the files attached by the user. The URLs are comma-delmited. **#!FileName[]!#**: returns the original file names associated with the files. **#!usermentions!#:** returns userid(s) of users mentioned by username or email. See Interpretation section for more details. **#!usermentionids!#:** returns usernames or useremails of existing users mentioned in the query. See Interpretation section for more details. **#!Interpret_Connections!#:** returns userid(s) of people that the person making the query may know and may have mentioned in the query. See Interpretation section for more details. In the previous example #!orderquantity!#, #!apples!#, #!mpa!#, #!ksi!#, and #!system!# are not system reserved variables. They are either variables defined in the input analysis stage or new variables created in the input processing stage. You may make if statements under return. If the statements are true, the system will return the variables under the condition.

api

There are two possible options: request-and-read, and request-only. The first supports REST and Querystring APIs and reads the output. The request-only option only sends the data to an end-point via POST and doesn't read the output.

request-and-read

api_base_url	The endpoint associated with the API
api_server_auth	Can be 0 or 1.
api_key_name	Specify the name of the key for authentication.
api_key_value	Specify the value of the key.
api_param_separator	Can be REST or QueryString.
api_parameters	List the parameters in the format below and define the default value for each parameter. Parameter name must match exactly the variable_name if you

	are reading data returned from the input processing stage. [{"action": "word-pronunciations","someparametername":"default value","word":"default word","language":"en", "userid": "0"}] You may define new parameters under the api_parameters. In the above example, the “language” parameter may or may not be a variable passed from input processing.
api_return_type	Can be xml or JSON.
return_key_hierarchy	Enter the hierarchy from parent to desired node separated by > For example: results>0>geometry>location
return_attributes	Specify the attributes that will be returned by the third party API. Must be comma-delimited. For example: Title, Description, Url.

request-only

apiURL	Specify the API end-point.
json_data	This is the payload that will be sent to your endpoint. Make sure it contains authentication information so you can verify that it’s coming from Nerddy. The formats of api_parameters of request-and-read and json_data of request-only are very different and can be confusing. In json_data, you specify the name of the parameter and then you may assign a static value or load a value from a parameter passed from input processing under it. Carefully observe the example below: {"appid":"xxxx", "authkey":"xxxx", "user":"#!userid!#", "fileurls":"#!File[]!#", "firstfilename":"#!FileName[1]!#","userqueryid":"#!userqueryid!#", "usersessionid":"#!usersessionid!#"} Data contained under the variables will be returned in place of the variables #!variablename!#

output-processing

The purpose of the output processing system is to process the data read by the system in API communication stage or input processing stage. This is an optional stage. Your app can use the Notification API to send asynchronous output to the user instead of returning output in the API communication stage.

A few things you should know:

• Output processing instructions must start with <output_processing> and end with </output_processing>.
• Output processing instructions must contain <output_display> </output_display> tags inside the <output_processing> tags.

• Output processing instructions may contain CSS in the beginning before the <output_display> tags and after the <output_processing> opening tag. CSS must be put under <css></css> tags. For example:

```
<css>

coolcat {

   background-color: linen;

}

</css>
```

• Use proper syntax as directed above; otherwise, the app instructions will not be parsed properly. The tags may seem unnecessary and excessive. However, they serve a purpose on our parsing side.

• The output display system supports if statements. For example:

```
if #!result[1][name]!# != NULL then{

display(!

<div class="clear2"></div>

Top Search Result: <b>#!result[1][name]!#</b>

<div class="clear1"></div>

!)}
```

• The most important function in the output display system is the display(! whatever !) function/block. Multiple display functions can be included under the output_display tags. Display functions can be included under if conditions. Display functions may contain for loops. Display statements should not be followed by a semicolon.

• You can print variables directly inside the display function by simply mentioning the variables.

For example: display(! Your age is #!age!# and your height is #!height!#cm !)

If #!age!# holds 7 and #!height!# holds 183, the system will return: Your age is 7 and your height is 183cm

• You may include HTML under the display function

• You may include Nerddy Mark-up for notifications under the display functions. Please refer to the Notifications API manual for more information of the mark-up. General rule: the output display system relies on the notification system. Whatever is allowed in the notification system is allowed in the output-processing display system.

• If the variable value contains spaces or special characters, sometimes it may be necessary to use ^enc as in #!variablename^enc!# to print the value of the variable with the special characters encoded. This is useful if you are printing variables to a URL to display some result in an iframe.

• Data received in the API communication stage is stored under #!result[order][attribute_name]!#

• For pagination, the system relies on the Notification API's pagination system. Therefore, if you wish to display content in different pages to the user, you must use the <page[j]></page[j]> tags.

```
display(!

<page[1]>This is page 1</page[1]>

<page[2]>This is page 2</page[2]>

!)
```

In the above example, the user will see "This is page 1" on the first page and "This is page 2" on the second page of the output.

• For loops in output display serve a limited purpose and that is to print #!result[order][attribute_name]!# arrays. They don't support nesting and can't contain variables from input processing. No more than one loop can be included inside one display block. For loops always follow this form: for (i=x, i<=y, i++). The middle operator can only be <= or < and the last part must be increasing (i.e. variable++).

• For loops can contain HTML and Nerddy Mark-up

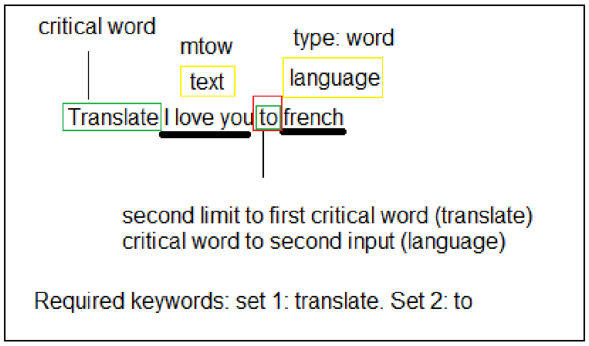
• For pagination, please observe the following example carefully:

```
display(!
for (j=1, j<=n/3, j++){
<page[j]>
for(i){
        <div class="clear2"></div>
        <NNB[i]>#!result[i][title]!#</NNB[i]>
        <div class="clear1"></div>
        Url: #!result[i][url]!#
        <div class="clear2"></div>
        <hr>
}
for(i){
        <NerddyNewBox[i]><center><iframe src="#!result[i][url]!#" width="900"
        height="600"></iframe></center></NerddyNewBox[i]>
}
</page[j]>
}
!)
```

In the above example, the developer wants to return web search results and uses Nerddy Markup so that when the user clicks on the website title, the webpage loads up in an iframe in a new box. The developer wants to return 3 results per page and doesn't know exactly how many results will be returned by the API and stored under the results array. Notice carefully the for(i) loop and the middle part of the main loop. The for(i) loop can be loaded inside normal for loops. In the middle part of the main loop, j<=n/3, n is the number of records in the result[order][attribute_name]. This means there will be as many pages as the number of records divided by 3 and rounded up. You can set it to any number and not necessarily 3. The for(i) loop automatically connect s to the main loop. Three new results will be printed each time the for(i) loop is called.

• JavaScript is not allowed

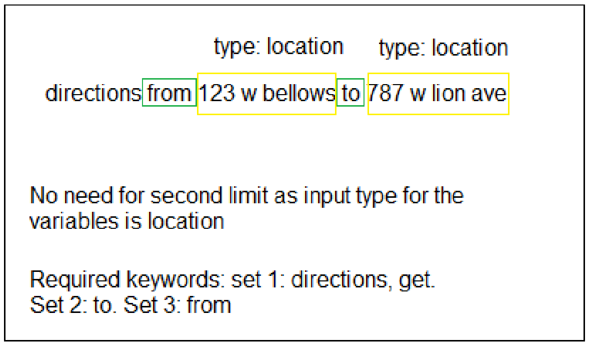
Interpretation and Analysis Examples



Translate I love you to french

second limit to first critical word (translate)
critical word to second input (language)

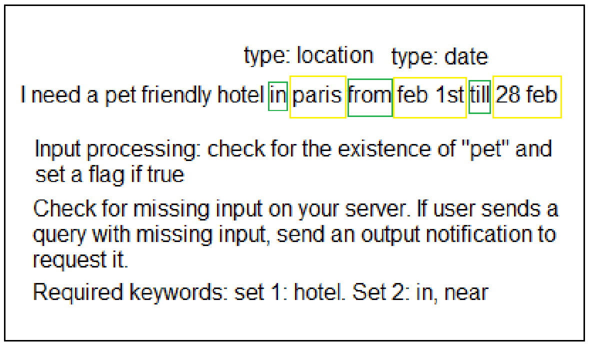
Required keywords: set 1: translate. Set 2: to



directions from 123 w bellows to 787 w lion ave

No need for second limit as input type for the variables is location

Required keywords: set 1: directions, get. Set 2: to. Set 3: from

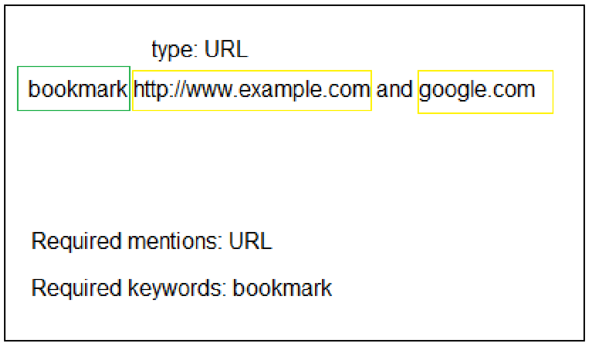


I need a pet friendly hotel in paris from feb 1st till 28 feb

Input processing: check for the existence of "pet" and set a flag if true

Check for missing input on your server. If user sends a query with missing input, send an output notification to request it.

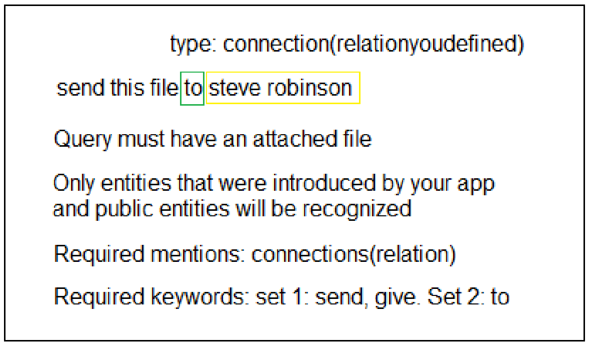
Required keywords: set 1: hotel. Set 2: in, near



bookmark http://www.example.com and google.com

Required mentions: URL

Required keywords: bookmark



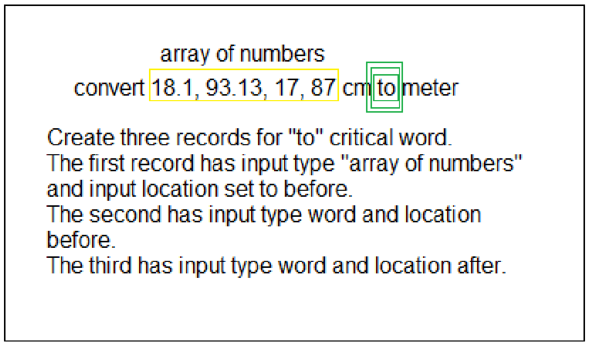
send this file to steve robinson

Query must have an attached file

Only entities that were introduced by your app and public entities will be recognized

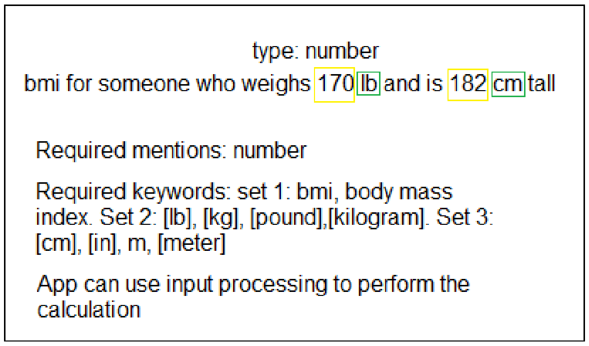
Required mentions: connections(relation)

Required keywords: set 1: send, give. Set 2: to



convert 18.1, 93.13, 17, 87 cm to meter

Create three records for "to" critical word.
The first record has input type "array of numbers" and input location set to before.
The second has input type word and location before.
The third has input type word and location after.

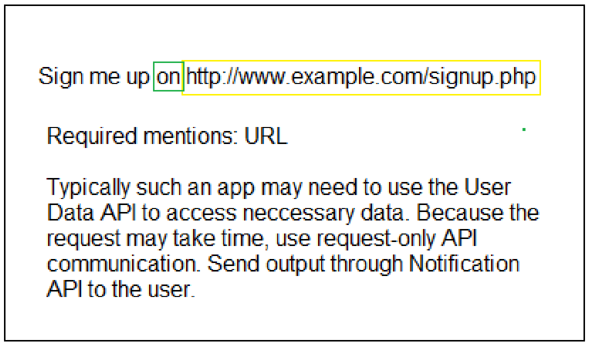


bmi for someone who weighs 170 lb and is 182 cm tall

Required mentions: number

Required keywords: set 1: bmi, body mass index. Set 2: [lb], [kg], [pound],[kilogram]. Set 3: [cm], [in], m, [meter]

App can use input processing to perform the calculation



Sign me up on http://www.example.com/signup.php

Required mentions: URL

Typically such an app may need to use the User Data API to access neccessary data. Because the request may take time, use request-only API communication. Send output through Notification API to the user.

🟨 variable 🟩 critical word 🟥 second limit

mtow: more than one word (input type)

Special Guidelines and Tips

• Don’t show your logo on the top left of the output; this is way too old fashioned. Use the attribution system for branding. Minimize the graphics in the output.

• Apps that use the wallet system must use the attribution system and offer customer support and terms pages.

• For more complex apps, it’s recommended that you use the POST API system to get the input and information you need and then you can send the output to the user asynchronously through the Notification API.

• You must develop a solid understanding of Nerddy’s interpretation and analysis systems to be able to build robust apps. Think of different ways users could word queries that they intend to reach your app.

• Don’t hesitate to contact the developer support if you have a product but unable to develop a Nerddy app for it.

• Nerddy Explore page should contain instructions to users such as queries supported by your app. Don’t create a Nerddy Explore page for each child app. Only create one page for the parent app.

• Use sidebar notifications to advertise your app to target audience.

• Advertise your Nerddy Explore page on social media. Use GIFs on Nerddy Explore articles to show users quickly how to reach your app.