

D8.4

System extension exhibiting serendipitous behaviour

Authors	Joseph Corneli, Simon Colton
Reviewers	Ewen Maclean

Grant agreement no.	611553
Project acronym	COINVENT - Concept Invention Theory
Date	September 1, 2015
Distribution	PU/RE/CO

Disclaimer

The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

The project COINVENT acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open Grant number 611553.

Abstract

The work presented here relies on the independent extension of individual COINVENT modules. A distributed, combinatorial search process seeks to find novel combinations of modules that produce output satisfying a success condition. This may occur for surprising reasons, for instance, if the outside world changes, or if the system gains new knowledge that unlocks a new approach.

Keyword list: **serendipity, tests, automated programming, flowcharts, type signatures.**

Changes

Version	Date	Author	Changes
0.1	01.08.15	Joseph Corneli	Creation
0.2	01.09.15	Joseph Corneli	Revision for internal review
0.3	07.09.15	Joseph Corneli	Minor corrections to text
0.3	30.09.15	Joseph Corneli	Address reviewer comments

Executive Summary

Rather than relying on a specific module that can be plugged into the system to introduce serendipitous effects, the work presented here relies on the independent extension of individual COINVENT modules. The system has the potential to realise “serendipity”, understood as a two-phase process involving the *discovery* of something unexpected in the world and the *invention* of an application for the same.

The flowcharting system FloWr provides a suitable basis for the system extension because of its intrinsic modularity, and its support for automated programming. Discovery is broken down into a *generative process* that produces combinations of nodes in new flowcharts and a *reflective process* that notices interesting which flowcharts are potentially interesting. In the prototype, flowcharts are potentially interesting either if they haven’t been tried before, or if they could be possibly be repaired through a simple modification. Invention is broken down into an *experimental process* and an *evaluation process*. In these early experiments, the basic result the system is aiming to achieve is simply to generate a new combination of nodes that can fit together and that generate non-empty output. The system has a “prepared mind” comprised of tests that tell it which nodes can fit together, as well as a record of previous trials. It queries FloWr via its Web API to test new combinations of nodes to see if they produce results.

Future work centres on making the system more discriminating, so that instead of simply searching for non-empty output, the process could search for output with some particular quality. Wrappers for COINVENT software components that are available as web services have been developed, which means they can be included in flowcharts. Improvements to search, and the possibility of blending flowcharts are interesting prospects for future work.

Contents

1	Introduction	1
2	Background: Features of the FloWr system	3
3	Overview of the system extension	4
4	Prototype code for testing node combinations	5
5	Future Work	6
6	Conclusions	7
Annex		
A	Tests that qualify node input and output	8
B	Prototype for a web-servicised COINVENT architecture	9

1 Introduction

Two key phrases need to be clarified at the outset: first, the meaning of *system extension* that this work assumes, which is straightforward, but requires some comment, and second, the particular interpretation of *serendipitous behaviour* that we make use of, which has been discussed in depth in D5.1 and D5.2, but which should be clarified here from the point of view of system building.

System extension Rather than relying on a specific module that can be plugged into the system to introduce serendipitous effects, the work presented here relies on the independent extension of individual COINVENT modules. Serendipity is not exhibited by one specific module, but is a global, aggregate behaviour of the system as a whole. What is described here is a route by which individual components of the COINVENT system can be extended with an interface to another system, in such a way that the resulting global behaviour will exhibit serendipitous effects.¹ Simplified examples will be discussed, and as more components of the system are extended, richer examples will become available.

Serendipitous behaviour Our understanding of the term “serendipity” as a two-phase process involving *discovery* of something unexpected in the world and *invention* of an application for the same is developed in detail in D5.1. In D5.2, evaluation standards for computational serendipity are described, and applied to three case studies. The key features of the model are the **trigger** T , which arises by *chance* outside the control of the system, the **prepared mind**, with components p and p' which, respectively, supply the *curiosity* required perform a **focus shift** – noticing that T is interesting – and the *sagacity* to find a **bridge** from the now-interesting trigger T^* to a **result**, R , with positive *value*. Figure 1 shows a relatively generic architecture diagram, which serves as a guide to the implementation work that will be discussed here. In this figure, p is viewed as an iterative process, divided into two components: p_1 , which notices particular aspects of the data, and p_2 , which offers reflections about those aspects. Similarly, p' is an iterative process that relies on functions p'_1 and p'_2 , which verify interesting features of the trigger by devising experiments and assessing their results.

The system to which individual modules are to be integrated is called FloWr [1]. FloWr is a user interface for creating and running *flowcharts* built of small modules called ProcessNodes, each of which carries out a simple processing step. Figure 2 is a screenshot showing an example of a flowchart. In day-to-day use, FloWr can be used a visual programming environment. It can also be invoked programmatically, on the Java Virtual Machine – or using any language via a new web API. The goals of FloWr are to be both a user friendly tool for co-creativity, and an autonomous *Flowchart Writer*.

Here, we target the latter scenario, and look at ways to assemble available ProcessNodes into flowcharts automatically, and evaluate the results. Some changes to FloWr have been necessary to facilitate this work, and these will be described, along with a preliminary evaluation of the system’s serendipity. FloWr provides a suitable basis for the system extension because of its intrinsic modularity, and its support for automated programming.

¹One can compare the notion of a *fibre* over a manifold.

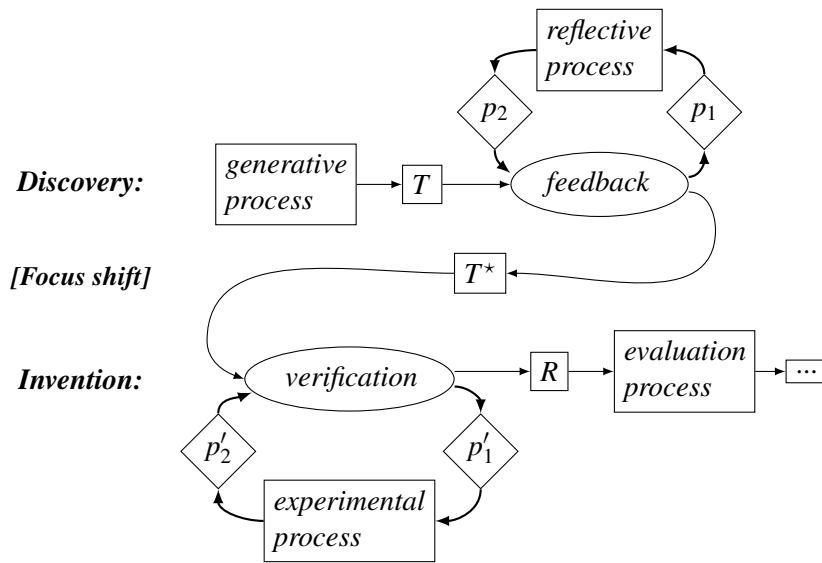


Figure 1: Generic architecture sketch for a serendipitous system

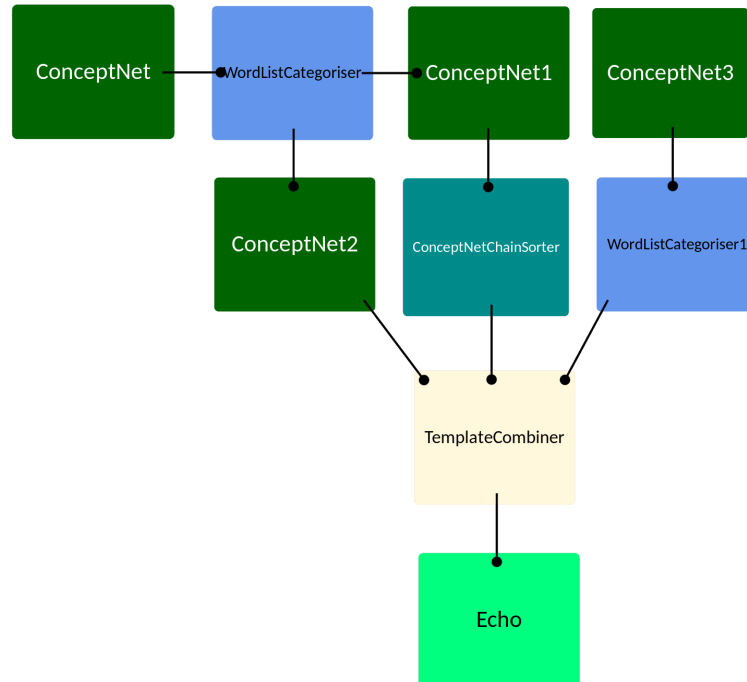


Figure 2: An example of a flowchart in the FloWr system

2 Background: Features of the FloWr system

In the backend, FloWr’s flowcharts are stored as scripts. These specify the names of the nodes involved in the flowchart, together with their (input) *parameters* and (output) *variables*. A connection between nodes is established when one node’s input parameter references the output variable of another node. Listing 1 is an example with two nodes. Here the `WordSenseCategoriser`’s `stringsToCategorise` parameter takes its setting from the `Dictionary`’s `#dictionaryWords` output variable.

```
text.retrievers.Dictionary.Dictionary_0
lowerFrequencyPercent:90
upperFrequencyPercent:95
numberRequired:10000
#dictionaryWords = words[*].text

text.categorisers.WordSenseCategoriser.WordSenseCategoriser_0
requiredSense:aj0
stringsToCategorise:#dictionaryWords
#adjectives = haveMainSense[*]
```

Listing 1: An short script comprised of a `Dictionary` and a `WordSenseCategoriser`

Inputs and outputs both come with constraints. For example:

- The `Dictionary`’s `lowerFrequencyPercent` and `upperFrequencyPercent` parameters are required to be between 0 and 100 and 1 and 100, respectively; and the former should be less than the latter.
- Both `#dictionaryWords` and `#adjectives` are guaranteed to be `ArrayLists` of strings, where each string is a single word.
- The `WordSenseCategoriser`’s `stringsToCategorise` parameter needs to be seeded with an `ArrayList` of strings. The node produces useful output only when these strings can be parsed as as a space-separated list of words.
- The `WordSenseCategoriser`’s `requiredSense` parameter needs to be seeded with a string that represents exactly one of the 57 British National Corpus Part of Speech tags.

Given constraints of this nature, the first challenge in automated flowchart assembly is to match inputs to outputs correctly, and to make sure that all required inputs are satisfied.

Appendix A lists new tests that have been introduced to guarantee that constraints of this nature are met. Node authors can use these tests to specify constraints on inputs and outputs (see Listing 2 for an example).² The more detail that can be added to these descriptions, the more subsequent users of the node can reason about its behaviour.

²The `Comment` formalism has been prototyped in FloWr, but is not yet active in the deployed system. As explained in Section 4, current experiments were conducted using the same information, copied to another codebase that integrates with FloWr over the web API.

In short, this additional information should be seen as part of the (*type*) *signature* of each FloWr node along with the standard Java type information (String, ArrayList[String], etc.). Strictly speaking, only some constraints, like integers that are required to fall within a given range, can be specified by subtyping; other more global constraints, such as relationships between parameters and variables, need more elaborate specifications.

```

public static String lowerFrequencyPercentComment = "S:Required;Test:  →
    IntInRange[0,100]";
public static String upperFrequencyPercentComment = "S:Required;Test:  →
    IntInRange[1,100],IntBiggerThan[lowerFrequencyPercent]";
public static String numberRequiredComment = "S:Required;Test:      →
    PositiveInteger";
public static String wordsComment = "S:Optional;Test:EachOne[IsWord]";
public static String wordsTuplesComment = "S:Optional;Test:      →
    EachOneInPlaces[tuplesPositionsToKeep,IsWord]";
public static String tuplesPositionsToKeepComment="S:RequiredWith[  →
    wordsTuples];Test:ExclamSeparatedIntsOrAll"

```

(a) Additional comment strings in Dictionary.java specify constraints on inputs

```


public static String lessFrequentWordComment = "Test:IsWord"
public static String mostFrequentWordComment = "Test:IsWord"
public static String wordsTuplesWithFrequency = "Test:MapSelectionFrom[  →
    wordsTuples]"
public static String words_textComment = "Test:IsWord"
public static String words_frequencyComment = "Test:FloatInRange      →
    [0,100]"
public static String words_totalOccurrencesComment = "Test:          →
    NonNegativeInteger"
public static String words_senses_senseComment = "Test:StringInList[  →
    CombineLists[BNC-POS-TAGS,BNC-POS-PTags]"
public static String words_senses_OccurrencesComment = "Test:        →
    NonNegativeInteger"

```

(b) Additional comment strings in DictionaryOutput.java specify constraints on outputs

Listing 2: Specifying the constraints on inputs and outputs for the Dictionary node

3 Overview of the system extension

The system is implemented following the general purpose design in Figure 1. The following text summarises the system features in terms drawn from the figure. Note that the system as a whole is part of a large-scale feedback loop; in other words, “” from Figure 1 feeds back into the generative process. This is explained in more detail below.

Generative process In our current experiment, the system’s potential **triggers** result from trial and error with flowchart assembly. Some valid combinations of nodes will produce results, and some will not. Due to the dynamically changing external environment (e.g., updates to data sources like Twitter) some flowcharts that did not produce results earlier may unexpectedly begin to produce results later on. The system will not try combinations that it knows cannot produce results, but it may retry earlier flowchart specimens that have the chance to become viable.

Reflective process The first aspect of the system’s **prepared mind** lies in a distributed knowledge base provided by ProcessNode signatures, and another crucial aspect its ability to test candidate flowcharts by running them. Randomly assembling a collection of nodes for which no known working combination existed into a flowchart that works is an occasion for a **focus shift**. The new working flowchart becomes interesting.

Experimental process This prompts the question: what made this particular combination work? Is there a pattern that could be exploited in the future? It may be that no broader pattern can be found, other than the fact that the combination works, in which case, the successful combination is the only tangible result, and it is recorded. Successful combinations and any further inferences about them form a third aspect of the system’s **prepared mind**. The **bridge** to the next set of results (new node combinations and new heuristics for assembling nodes) is accordingly found by informed trial and error.

Evaluation process In these early experiments, the basic **result** the system is aiming to achieve is simply to generate a new combination of nodes that can fit together and that generate non-empty output. Subsequent versions of the system may have more detailed evaluation functions, setting a higher bar. For example, a future version of the system could be tuned to search for flowcharts that generate meaningful poems, as we discuss in [2].

4 Prototype code for testing node combinations

The current code for developing, testing, and reasoning about combinations of nodes is written in Python. Since potential node combinations can be computed using only the node’s signature, information about node signatures has been extracted and stored separately from FloWr. The system queries FloWr via the Web API only to test a new combination of nodes to see if it produces results. It includes an integration with Java using PyJNIus³ that makes the tests described in Appendix A available in Python, so that potential inputs and received outputs can be tested locally.

A successful combination serves as a template that can be instantiated later. Reasoning backwards from a particular goal (e.g., a specific type of output with particular qualitative features) is supported at a rudimentary level. Successful combinations that can produce a particular kind of output are stored. This is useful in subsequent runs, in which that type of output may be needed as an input.

The code for the current prototype is available at <https://github.com/holtzermann17/FloWrTester>. Note, a login to FloWr is required in order to obtain an up-to-date API key, which needs to be copied into the code.

³<https://github.com/kivy/pyjnius>

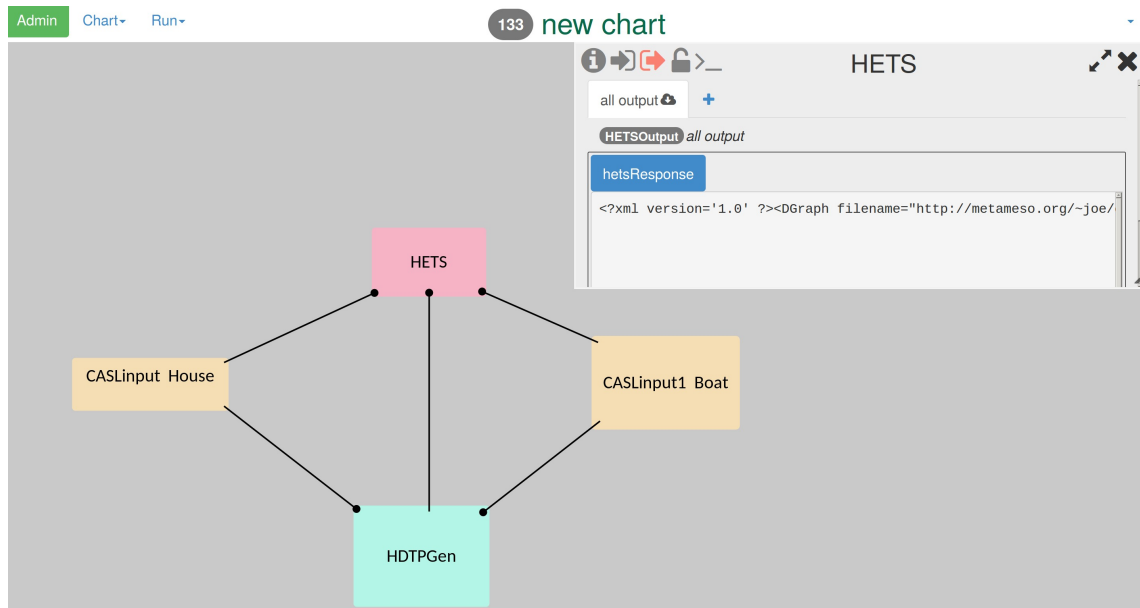


Figure 3: Basic demo of COINVENT component integration, using FloWr's web interface

5 Future Work

The more discriminating the system can be about measures of value, the more meaningful an attribution of serendipity to any particular successful run will be. One interesting route would be to assess the value of explanatory heuristics, rather than generated texts; this would require increased sagacity on the part of the system. More stringent goals would inform this process. For example, instead of simply searching for non-empty output, the process could search for output with some particular quality (e.g., a subject/verb/object format). FloWr can already reliably generate output of this sort, using flowcharts that have been built by hand. Interaction between different heuristically-driven search processes would also be possible, and could produce more surprising results. Again in the poetry context, one search process could look for narrative outlines that would structure a poem with, and another process could look for lines or stanzas to fill out that outline.

The population of ProcessNodes constrains (and, as more nodes are added, extends) the possible strategies for assembling flowcharts. New nodes that carry out core functions of the COINVENT system will make outputs and reasoning steps relevant to the core domains of mathematics and music available to the system. Appendix B gives an example of a ProcessNode built around a generic webservice. Writing wrappers for COINVENT components that have been turned into webservices is similarly straightforward, and has been prototyped (Figure 3). With more fine-grained modules, a more targeted combinatorial search for new solutions can be conducted. For instance, modules that could combine to generate valid CASL input would allow the system to search over the space of valid theories.

The concept of search can be introduced into flowcharts themselves at various levels. Indeed, at each point in a flowchart there is, in principle, the possibility to run a search. As more options are generated, it may be relevant to backtrack and select a different part of the search space to explore. FloWr can support simple loops, but more complex loops that would adjust the structure

of underlying programs (as envisioned in Figure 1) are not yet supported. Valuable search parameters that are given a favourable local evaluation may also be propagated forward, as we examined in a distributed model of search that aimed to realise a global condition [3].

Given the importance of signatures in the COINVENT formalism, the work presented here suggests the interesting possibility of blending flowcharts. This has not been attempted; however in [4], we examine the somewhat related issue of blending succinct video game descriptions, written in the Video Game Description Language (VGDL). An ability to blend objects from semantic domains not initially within the COINVENT remit would show the strength of the approach.

6 Conclusions

Describing the relationships between nodes that exist in the form of constraints allows automated processes to assemble components into functioning programs. Explicitly specifying component interfaces supports a combinatorial search for new solutions. Happy, unexpected, accidental discoveries are more likely to occur, and to register, if, in addition, we are able to discern the properties and value of generated artefacts. Combinatorial and evaluative concerns can both be addressed with programmatic tests. The expressiveness of the testing and constraint language is a major limitation; another limitation is the size of the collection of available ProcessNodes. Current work is aimed at facilitating further extensions. Where possible, input fields should be broken down into machine-understandable components. The result will be a version of the COINVENT system that can be used by programs as well as by people, and that can (sometimes) generate serendipitous results.

References

- [1] COLTON, S., AND CHARNLEY, J. Towards a Flowcharting System for Automated Process Invention. In *Proceedings of the Fifth International Conference on Computational Creativity* (2014), D. Ventura, S. Colton, N. Lavrac, and M. Cook, Eds.
- [2] CORNELI, J., JORDANOUS, A., SHEPPERD, R., LLANO, M. T., MISZTAL, J., COLTON, S., AND GUCKELSBERGER, C. Computational poetry workshop: Making sense of work in progress. In *Proceedings of the Sixth International Conference on Computational Creativity, ICCO 2015*, S. Colton, H. Toivonen, M. Cook, and D. Ventura, Eds. Association for Computational Creativity, 2015.
- [3] CORNELI, J., AND MACLEAN, E. The search for computational intelligence. In *Social Aspects of Cognition and Computing Symposium, Proc. Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (SSAISB), University of Kent, Canterbury, UK, 20-22nd April 2015* (2015), Y. J. Erden, R. Giovagnoli, and G. Dodig-Crnkovic, Eds.
- [4] GOW, J., AND CORNELI, J. Towards generating novel games using conceptual blending. In *Proceedings of the Second AIIDE Workshop on Experimental AI in Games (EXAG2), 14-15 November 2015, Santa Cruz, CA, USA* (2015), M. Cook, A. Liapis, and A. Zook, Eds. To appear.

A Tests that qualify node input and output

IsRegex (String) Guarantees that the object is a Java regular expression.

StringInList (String, String[]) Guarantees that the given string is an element of the given list of strings.

PositiveInteger (int) Guarantees that the given integer is positive.

NonNegativeInteger (int) Guarantees that the given integer is ≥ 0 .

IntAsString (String) Guarantees that the string can be parsed as an integer.

IntAsStringOrAll (String) Guarantees that the string can either be parsed as an integer or else that it is the unique string “all”.

IntInRange (int, int, int) Guarantees that the first argument is between the other two (inclusive).

FloatInRange (float, int, int) Guarantees that the first argument is between the other two (inclusive).

IntLessThan (int, int) Guarantees that the first argument is less than the second argument (exclusive).

IntBiggerThan (int, int) Guarantees that the first argument is bigger than the second argument (exclusive).

IntLeqThan (int, int) Guarantees that the first argument is less than the second argument (inclusive).

IntGeqThan (int, int) Guarantees that the first argument is bigger than the second argument (inclusive).

IsWord (String) Guarantees that the argument does not contain any spaces.

ExclamSeparatedWords (String) Guarantees a `!!`-separated list of words.

ExclamSeparatedInts (String) Guarantees a `!!`-separated list of integers, as strings.

ExclamSeparatedIntsOrAll (String) Guarantees a `!!`-separated list of integers, as strings, or “all”.

SemicolonSeparatedWords (String) Guarantees a `;`-separated list of words.

ExclamSeparatedItemsFromList (String, String[]) Guarantees that the input string is a `!!`-separated list of items from the string array.

IntMinimizesTuplesLengths (int, ArrayList<String[]>) Guarantees that the input integer is smaller than the length of the shortest input tuple.

UnderscoreSeparatedWords (String) Guarantees a `_`-separated list of words.

EachOne (String, ArrayList<String>) Guarantees that the test named by the input string is satisfied by each element of the ArrayList of strings.

B Prototype for a web-servicised COINVENT architecture

The following code gives an example of a web-service wrapped up as a `ProcessNode`. It queries the public AlchemistAPI service for JSON with a GET request, and uses that to set an output variable. Querying web services using a POST request is similar.

Alchemist.java:

```
package ccg.flow.processnodes.mynodefolder.Alchemist;

import ccg.flow.processnodes.ProcessNode;
import ccg.flow.processnodes.ProcessOutput;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.HashMap;
import java.util.Map;

import java.net.MalformedURLException;
import java.io.IOException;

import org.json.simple.*;
import org.json.simple.parser.JSONParser;

// other JSON libraries mentioned in the comments at
// http://stackoverflow.com/a/18998203/821010
// are likely to be more convenient

public class Alchemist extends ProcessNode {

    // according to the service provider this should be kept secret
    // it is easy to get your own from http://www.alchemyapi.com/
    public String apiToken = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
    public String apiEmail = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx";
    public String apiCommand = "calls/data/GetNews";

    @Override
    public ProcessOutput process () {

        AlchemistOutput output = new AlchemistOutput();

        String url = "http://access.alchemyapi.com/";
        String charset = "UTF-8";

        // the required parameters will go into the URL
        Map<String,String> params = new HashMap<String,String>();
```

```

params.put("apikey", apiToken);
params.put("outputMode", "json");
params.put("start", "now-1d");
params.put("end", "now");
params.put("maxResults", "100");
params.put("q.enriched.url.enrichedTitle.relations.relation", "|
    action.verb.text=acquire,object.entities.entity.type=Company
|");
params.put("return", "enriched.url.title");

params.put("c", apiCommand);

try {
    StringBuilder sb = new StringBuilder();
    for (Map.Entry<String, String> entry : params.entrySet()) {
        if (sb.length() > 0) sb.append('&');
        sb.append(URLEncoder.encode(entry.getKey(), charset) + "="
            + URLEncoder.encode(entry.getValue(), charset));
    }

    String query = new String(sb);

    // route the query to the URL via the correct API command
    HttpURLConnection con = (HttpURLConnection) new URL(url +
        apiCommand + "?" + query).openConnection();
    con.setRequestMethod("GET");
    con.setDoOutput(true);
    con.setRequestProperty("Accept-Charset", charset);
    con.setRequestProperty("Content-Type", "application/x-www-
        form-urlencoded; charset=" + charset);

    System.out.println("\nSending 'GET' request to URL : " + url
        + apiCommand + "?" + query);
    System.out.println("Response Code : " + con.getResponseCode
        ());

    InputStream is = con.getInputStream();
    BufferedReader in = new BufferedReader(new InputStreamReader
        (is));
    String s;

    StringBuilder response = new StringBuilder();
    while ((s = in.readLine()) != null)
        response.append(s);
    in.close();

    String responseAsString = new String(response);

    // Parse the JSON response

    Object obj = JSONValue.parse(responseAsString);

```

```
JSONObject jobj = (JSONObject)obj;
JSONObject result = (JSONObject)jobj.get("result");
JSONArray docs = (JSONArray)result.get("docs");

String titles = "";

for (int i = 0; i < docs.size(); i++)
{
    JSONObject doc = (JSONObject)docs.get(i);
    JSONObject source = (JSONObject)doc.get("source");
    JSONObject enriched = (JSONObject)source.get("
        enriched");
    JSONObject docurl = (JSONObject)enriched.get("url");
    String title = (String)docurl.get("title");
    titles += title + "\n";
}

// populate the output
// (in this case, just with the concatenated titles,
// in general we can build something with more structure)
output.returnValue = titles;
}
catch (Exception ex) {
    if (ex instanceof MalformedURLException) {
        reportError("Malformed URL.");
        return null;
    } else if (ex instanceof IOException) {
        reportError("Unable to open connection: " + ex.getMessage
            ());
        return null;
    }
}

return output;
}
}
```

AlchemistOutput.java:

```
package ccg.flow.processnodes.mynodefolder.Alchemist;

import ccg.flow.processnodes.ProcessOutput;

public class AlchemistOutput extends ProcessOutput {

    // Just a string for this demo
    public String returnValue;

}
```