**Inkwell C++ API Documentation**

**English**

****

**Table of contents**

1. About the project 3
2. How to include the API 4
3. API Classes. Fields and methods 5
   1. Project 5
   2. Table 7
   3. Entry 8
      1. Event 9
      2. Fact 10
      3. Rule 11
         1. Criterion 12
         2. Modification 13
4. API Interface 14
5. API Errors and Error Handling 16
   1. All error messages and their meaning 16

This document will attempt to cover everything regarding the Inkwell API.

About the project

Inkwell is a project aimed towards game developers. It attempts to make the story implementation process easier by structuring information in three different Entry types, as such allowing the development team to not have to focus on the process of implementing the story in code form. This process is very likely to become messy as the story expands. The Inkwell API attempts to provide an easy to use and robust interface that brings great flexibility to the story development task.

This project utilizes an external JSON parsing library. While this choice makes the parsing process slightly less efficient, we believe that it greatly increases the safety aspect of the parsing. By using a solid and well-tested library for this step, we are ensuring the API performs properly and responds to faulty input, avoiding silent runtime errors.

We believe the idea behind the application is rather clever and it can significantly improve the development process of any story-driven game. As a practical example to this statement, the video game Undertale by Toby Fox, a game entirely reliant on its storytelling, had its main selling characteristic built on a rather messy foundation, which hindered the development process greatly. As such, the follow-up game, Deltarune, had its story structured in an expandable and organized manner that is very similar to the one that Inkwell is aiming for.

The idea behind the Inkwell system is not entirely ours.

It was pitched during a GDC talk: “Do You Copy? Dialog System and Tools in Firewatch” (https://www.youtube.com/watch?v=wj-2vbiyHnI).

Another GDC talk that promotes this idea is “Programming Context-Aware Dialogue in The Last of Us” (<https://www.youtube.com/watch?v=Y7-OoXqNYgY>).

We are the first people to attempt to implement this sort of system in a manner that is open-source, free for everybody to use and easy to work with.

**How to include the API**

Download the project (from Github or otherwise).

Go to the downloaded ZIP file, to the src folder, and then the include folder within. Copy the include folder and paste it into your project.

Once done, in the place where it is needed within the project, write:

#include <include/inkwell.hpp>

The include folder is structured as such:

include

* inkwell
  + inkwellEntry.hpp
  + inkwellEnums.hpp
  + inkwellErrors.hpp
  + inkwellProject.hpp
  + inkwellScope.hpp
  + inkwellTable.hpp
* nlohmann
  + json.hpp
* inkwell.hpp

**API Classes. Fields and methods**

* **Project**

Please note that the Project object is meant to be used in shared pointer form, as such:

std::shared\_ptr<Project> name = std::make\_shared<Project>();

Using it in non-pointer form may possibly cause unexpected issues!!

The Project is the main part of the API and the base object that the user will interact with.

* Project Class contents:

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| Project() | Default constructor |
| Project(int flags) | Constructor with flags |
| getFlags() | Returns project flags (int) |
| getExtraData() | Returns extra data about the project. This extra data is represented by information that the Inkwell web application uses for certain features. That is, anything outside of the tables and scopes fields within the JSON. |
| INIT() | Returns whether project is initialized (boolean) |
| T(std::string key) | Returns shared pointer to table with matching key.  If no key matches, an exception will be thrown. |
| T(int id) | Returns shared pointer to table with matching ID.  If no ID matches, an exception is thrown. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| initialized | Private | Boolean that states whether Project is initialized. Is set to true once the >> operator is used on the Project. This prevents the Project from being parsed twice. |
| idToKey | Private | Int-String map used internally to turn IDs into Keys. |
| tables | Private | String-Table map used internally for the T() function. |
| extraData | Private | Json object used to store the extra data regarding the Project (data that is not relevant in the API, but the Inkwell web site). |
| flags | Private | Int that stores flags regarding the Project. Currently there are only flags that state whether certain extra information will be parsed from the file and into the extraData variable. Set to 0 by default (none). |
| tabIndex | Private | Int used internally for the parsing process. |

1. Flags

|  |  |
| --- | --- |
| ***Flag***  ***(PROJECTFLAG\_)*** | ***Description*** |
| NONE (0) | No flags set |
| NOCLOUD (1) | cloud section will not be parsed from file |
| NOENTRYMAP (2) | entryMap section will not be parsed from file |
| NOINVITECODE (4) | inviteCode section will not be parsed from file |
| NOOWNER (8) | owner section will not be parsed from file |
| NOBANNER (16) | projectBanner section will not be parsed from file |
| NOCREATEDAT (32) | projectCreatedAt section will not be parsed from file |
| NODESCRIPTION (64) | projectDescription section will not be parsed from file |
| NOID (128) | projectId section will not be parsed from file |
| NONAME (256) | projectName section will not be parsed from file |
| NOMEMBERS (512) | members section will not be parsed from file |
| NOEXTRAINFO (1024) | All of the above flags |

* **Table**

The Table represents a container that holds Entries.

Tables can only be constructed internally (the user can not construct Table objects).

* Table Class contents:

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| ID() | Returns table ID |
| KEY() | Returns table Key |
| SC() | Returns table Scope |
| INIT() | Returns whether Table is initialized |
| E(std::string key) | Returns Event with specified Key. If no such Event exists, an exception is thrown. |
| E(int id) | Returns Event with specified ID. If no such Event exists, an exception is thrown. |
| F(std::string key) | Returns Fact with specified Key. If no such Fact exists, an exception is thrown. |
| F(int id) | Returns Fact with specified ID. If no such Fact exists, an exception is thrown. |
| R(std::string key) | Returns Rule with specified Key. If no such Rule exists, an exception is thrown. |
| R(int id) | Returns Rule with specified ID. If no such Rule exists, an exception is thrown. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| initialized | Private | States whether a Table is initialized. It is set to true when its constructor is called. Prevents certain fields from being modified once true. |
| id | Private | The ID of the table. (int) |
| key | Private | The Key of the table. (std::string) |
| scope | Private | The Scope of the table. |
| idToKey | Private | Int-String map used internally to turn IDs into Keys. |
| events | Private | String-Event map used internally, used by the E() function. |
| facts | Private | String-Fact map used internally, used by the F() function. |
| rules | Private | String-Rule map used internally, used by the R() function. |

* **Entry**

An Entry is an abstract class. It is inherited by three other classes: Event, Fact and Rule.

All Entries can only be constructed internally.

* Entry Class contents:

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| INIT() | Returns whether entry is initialized |
| ID() | Returns entry ID |
| KEY() | Returns entry Key |
| GRD() | Returns entry Guard (currently redundant) |
| virtual trigger() | Triggers the entry. Each of the three Entry types overrides this function. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| initialized | Private | States whether an Entry is initialized. It is set to true when its constructor is called. Prevents certain fields from being modified once true. |
| id | Private | The ID of the entry |
| key | Private | The Key of the entry |
| guard | Private | The Guard of the entry (currently redundant) |
| value | Public | The value of the Entry (double). This variable has a different meaning for each inheriting Entry type. |

* **Event**

An Event is an Entry type whose role is to trigger other Entries.

The Value field of an Event represents how many times its trigger() method has been called.

* Event Class contents (not inherited):

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| trigger() override | Triggers the entries in its triggers set. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| triggers | Private | Unordered set that contains pointers to all the Entries to be triggered upon a trigger() call. |

* **Fact**

A Fact is an Entry whose role is to hold data.

The Value field of a Fact represents its data.

Facts can not be triggered.

* Fact Class contents (not inherited):

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| trigger() override | Throws an exception, as Facts can not be triggered. |

1. Facts do not introduce any new Fields.

* **Rule**

A Rule is an Entry whose role is to dispatch callbacks, as well as modify and trigger other Entries.

The Value field of a Rule represents how many times it has successfully been triggered.

* Rule Class contents (not inherited):

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| trigger() override | Checks whether all its criteria are met. If not, it returns false. Else, it calls the modify() method of all its Modification objects, dispatches all its callbacks and triggers all the Entries in its trigger set, then returns true. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| triggers | Private | Unordered set that contains pointers to all the Entries to be triggered upon a trigger() call. |
| criteria | Private | Vector that contains pointers to all Criterion objects whose check() method will be called upon a trigger() call. |
| modifications | Private | Vector that contains pointers to all Modification objects whose modify() method will be called upon a trigger() call. |
| callbacks | Public | Vector that holds every callback function that will be dispatched upon a trigger() call. |

* **Criterion**

A Criterion has the role of checking whether an Entry’s value meets some condition.

Criteria can only be constructed internally.

* Criterion Class contents:

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| INIT() | Returns whether the criterion is initialized. |
| check() | Checks whether the specified Entry’s value meets the set condition. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| initialized | Private | States whether a Criterion is initialized. It is set to true when its constructor is called. Prevents certain fields from being modified once true. |
| comparedEntry | Private | The Entry whose value is compared during a check() call. |
| comparedEntryID | Private | The compared Entry’s ID. The point of this variable is to allow the Entry pointer to be linked after all Entries have been parsed in order to ensure correct linking. |
| compareValue | Private | The value which the Entry is compared against during a check() call. |
| comparisonOperator | Private | Dictates how the Entry will be compared: equal to, not equal to, less than, greater than, less than or equal, greater than or equal. |

* **Modification**

A Modification has the role of modifying Entries.

Modifications can only be constructed internally.

* Modification Class contents:

1. Public Methods

|  |  |
| --- | --- |
| ***Method*** | ***Description*** |
| INIT() | Returns whether the modification is initialized. |
| modify() | Modifies the Entry according to the supplied parameters. |

1. Fields

|  |  |  |
| --- | --- | --- |
| ***Field*** | ***Access specifier*** | ***Description*** |
| initialized | Private | States whether a Modification is initialized. It is set to true when its constructor is called. Prevents certain fields from being modified once true. |
| modifiedEntry | Private | The Entry which will be modified during the modify() call. |
| modifiedEntryID | Private | The modified Entry’s ID. The point of this variable is to allow the Entry pointer to be linked after all Entries have been parsed in order to ensure correct linking. |
| modificationOperator | Private | Dictates how the Entry will be modified: set to or incremented. |
| modifyWithValue | Private | Variable which will be utilized to modify the specified Entry’s value. |

**API Interface**

* Projects may be declared in shared\_ptr form, as mentioned before:

std::shared\_ptr<Project> project = std::make\_shared<Project>(myFlags);

* One may parse a project from any input stream by using the >> operator, just like any other variable:

myInputStream >> project;

* Access a Project’s Table by using either of the two T() functions:

project->T(“myTableKey”)

or

project->T(tableID).

* Access a Table’s Event, Fact or Rule by using the corresponding E(), F() or R() function:

project->T(“myTableKey”)->E(“myEventKey”)

project->T(tableID)->F(“myFactKey) etc.

* Here are some examples of accessing fields and methods:

1. A callback lambda function being added to the rule with the key “foo” in the table with the ID 8:

project->T(8)->R(“foo”)->callbacks.push\_back(

[](){std::cout << “Hello world!\n”; }

);

1. Checking whether the project is initialized:

if(project->INIT())

std::cout << “The project is initialized!\n”;

else

std::cout << “The project is NOT initialized!\n”;

1. Triggering an entry (in this case, an Event) and checking whether it was successful:

if(project->T(“tableeeess”)->E(11)->trigger())

std::cout << “It was triggered successfully!\n”;

* To format all of the Project’s data back into text file format, use the << operator (This operation also saves all of the values of the Entries etc.)

myOutputStream << project;

**API Errors and Error Handling**

All exceptions that are thrown by Inkwell are preceded by [Inkwell Error].

|  |  |
| --- | --- |
| ***Exception*** | ***Extra information*** |
| This (object) ((fields) (…)) has already been initialized! | The object has already been parsed and/or constructed, from a file or otherwise. |
| A (object) with the (fields) (…) does not exist! | An object with the specified identification data was not found in the Project. |
| A (object) with the (fields) (…) already exists! | The parser attempted to add an object whose identification data overlaps with another present object’s data. That data (The ID or the Key) is meant to be unique. |
| Invalid Comparison Operator (…) ! | The Comparison Operator provided to a Criterion is invalid. |
| Invalid Modification Operator (…) ! | The Modification Operator provided to a Modification is invalid. |
| Facts cannot be triggered! | An Entry has attempted to trigger a Fact. |
| String (…) is not recognized by the Enum Converter! | The parser has attempted to convert an invalid string into an Enum Key. |
| A key is not recognized by the Enum Converter! | An error occurred in the enum conversion process. |
| No entry with the (fields) (…) was found! | Either during the parsing process or when a getter function was called, no entry with the specified identification data was found in the Table. |