Charon Documentation

version 2025.1.3

Denis Zykov

March 27, 2025

Contents

Overview

Charon is a powerful data-driven game development tool designed to streamline the creation and management of static game data within your game. It allows both developers and game designers to efficiently model and edit game entities such as characters, items, missions, quests, and more, directly within the Unity/Unreal Engine/Browser environment. Charon simplifies the process of data manipulation, offering a user-friendly interface and automatic source code generation, which significantly reduces development time and minimizes manual coding errors. Charon also offers support for working with text in multiple languages, with easy loading and unloading of translated text.

With Charon, game developers can focus on creating engaging gameplay experiences without worrying about the technical details of managing game data. It is available in three deployment variants, including a standalone application, web application, Unity plugin and Unreal Engine plugin.

TLDR Charon is an in-game database for your game, replacing spreadsheets or config files.

Why Choose Charon?

Charon replaces traditional spreadsheets or config files with an in-game database, offering a structured and efficient way to manage game data. It allows developers to focus on creating engaging gameplay experiences without worrying about the technical details of data management.

Is It Free?

The offline version, CLI and plugins are completely free and have no restrictions. They are distributed under a free license and allow you to distribute tools along with the game for modding games.

The online version, which allows working in large teams on shared game data, requires a subscription.

What is Charon

It is a .NET 8 console application that can be used as a command-line tool for performing CRUD operations with your game data, or as an HTTP Server to provide a UI for modeling and editing your game data. There are plugins for Unity and Unreal Engine that provide a more integrated experience while using Charon. As with any modern .NET application, it can be launched as is on Windows, macOS and Linux and via *dotnet* tool.
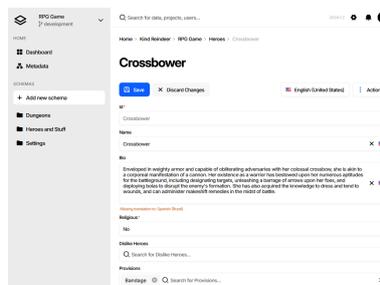
Further reading

Unreal Engine Plugin Overview

Charon is a versatile plugin tailored for Unreal Engine, designed to facilitate data-driven game design by allowing both developers and game designers to efficiently manage static game data, like units, items, missions, quests, and other. Unlike Unreal Engine's DataTables, Charon elevates the experience by offering an integrated editing UI directly within Unreal Engine, enabling the modeling of diverse data structures suited to any game genre. It provides a user-friendly interface that requires no special skills for game designers, simplifying the process of data manipulation. For programmers, Charon streamlines development workflows by generating code to load game data seamlessly into the game.

Key Features

- **Data Modeling**: Define game entities like characters, items, missions, quests, and dialogs to meet the specific needs of your game. Interconnect and fill these tables within one UI.

- **Error Control**: Implements validation checks to verify the accuracy of input data, reducing the likelihood of errors that could impact gameplay or development.

- **Code Generation**: Automates the creation of boilerplate code needed to work with your game data, significantly speeding up development time and reducing manual coding errors.

- **Spreadsheet Export/Import**: Offers seamless integration with spreadsheet software, enabling you to effortlessly populate, edit, and manage your game data in a familiar environment.

- **Localization Export/Import**: Simplifies the process of preparing game data for translation, making it straightforward to adapt your game for global audiences.

- **Modding Support**: Empowers your gaming community by providing them with the tools to create and share mods, enhancing the longevity and depth of your game.

- **Dynamic Load**: Facilitates the dynamic loading of game data, enabling features like A/B testing or the ability to push hot updates directly to your players.

Getting Started

To begin using this plugin, the initial step involves installing the plugin from the Unreal Engine Marketplace. Once installed, you'll need to enable the plugin for your project through the project settings. Following this, a rebuild of your project's C++ code is necessary. The final step in the setup process is the creation of your first game data file.

Prerequisites

The Unreal Engine plugin is written in C++ but relies on `dotnet charon`, a .NET Core application which runs on .NET 8.

Windows

1. Download and install NET 8+.

2. Make sure you have write access to `%APPDATA%/Charon`.

MacOS

1. Download and install NET 8+.

2. Make sure you have write access to `~/Library/Application Support/Charon`.

3. Make sure `dotnet` is available from `$PATH`.

Linux

1. Download and install NET 8+.

2. Make sure you have write access to `~/.config`.

3. Make sure `dotnet` is available from `$PATH`.

Checking Available .NET Versions

```
# check for dotnet already installed
dotnet --list-sdks
```

Installation from Marketplace

1. Add to cart Charon plugin [Epic Launcher] / [Web] in the Unreal Engine Marketplace.

2. Follow the instruction on installing plugin into your project:

   a. Click **Install to Engine** and select the engine version.

   b. Open your project and go to **Edit → Plugins…** window.

   c. Type **Charon** in the **Search** bar.

   d. Check the checkbox near the plugin's name to enable it.

3. Rebuild project C++ code.

Building from Source Code

1. Clone or download the plugin source code from the GitHub repository.

2. Create a `<project-dir>/Plugins/Charon` directory.

3. Copy the plugin files into this directory. Ensure **Charon.uplugin** is located at the path `<project-dir>/Plugins/Charon/Charon.uplugin` after copying.

4. Remove the `"EngineVersion"` attribute if your engine doesn't match the plugin's engine version.

5. Rebuild the project's C++ code.

6. Enable the plugin in **Edit → Plugins…** if needed.

Core Concepts

Data-Driven Design Principles

Data-driven design emphasizes the control of gameplay through data, rather than source code/blueprints, with game mechanics and processes determined by structured data files. For instance, rather than embedding damage calculations directly in the game's source code, these are defined by data specifying weapon effects and the rules for their application. Or for example, mission progression is not hardcoded; it's outlined in editable text files, making these aspects of game design highly flexible. This approach not only facilitates quick adjustments during development but also simplifies adding modding support post-release.

- Data Driven Gameplay Elements (UE Documentation)
- Modify Everything! Data-Driven Dynamic Gameplay Effects on 'For Honor' (Video)
- Data-driven Design in Unreal (Article)

Understanding the Plugin's Architecture

Plugin Assets

Working with data in this plugin is akin to how the built-in *DataTable* functions. There is a data source file, a module containing the code required to load the data, and an asset that will be utilized in the game. Whenever you edit a data source file, you need to re-import this data into the asset. Should the data structure in the source file change, then the C++ code must be regenerated.

For scenarios requiring dynamic loading of game data, this can be accomplished through the `TryLoad` method on the game data class, which accepts the source JSON file.

Plugin Modules

The Charon plugin is structured into two modules:

- `CharonEditor` module acts as an Unreal Engine Editor extension. Extension points for the module are declared in the `ICharonEditorModule` class, and automation of game data processing is facilitated through the `FCharonCli` class.

- `Charon` module, houses the core logic and shared code crucial for handling game data files.

Working with the Plugin

Creating Game Data

To create a new game data file within the Unreal Engine Editor, open the **Content Drawer**, right-click in the desired folder, and select in the **Create Advanced Assets** section **Miscellaneous** → **Game Data** menu option. Name your game data file and proceed according to the instructions in the dialog window that appears.

Detailed guide on how to create game data.

Editing Game Data
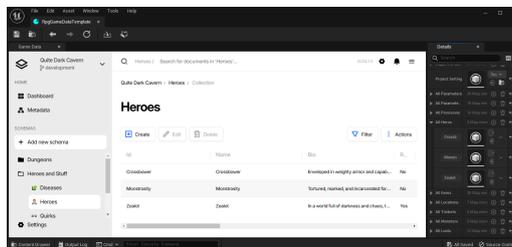
To edit a game data file in the Unreal Engine Editor, navigate to the **Content Drawer**, find the corresponding .uasset file, and double-click it. This action opens a new window featuring a user interface for editing the game data. Remember to reimport and, if necessary, regenerate the source code after completing your edits.

Refencing Game Data in Blueprints

Similar to the DataTable's `FDataTableRowHandle`, the Charon plugin introduces a specific type for referencing documents within Blueprints, named `FGameDataDocumentReference`. This type is housed within the Charon module. Here is example of **Game Data Document Reference** used to resolve *Hero* document:
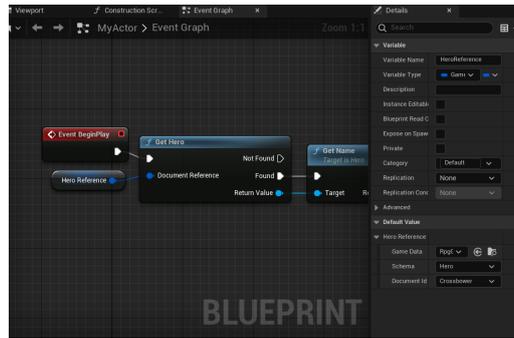
Advanced Features

Localization and Multi-Language Support

Charon facilitates multi-language text support through the `Localizable Text` data type. When creating a *Schema*, properties can be defined with various data types, including `Localizable Text`. Initially, all localizable text defaults to `EN-us` (US English). Additional languages can be added via **Project Settings** → **Internationalization** → **Translation Languages** in the Charon UI.

Exporting/importing localizable data.

Referencing Unreal Engine Assets

By default, game data files and the Charon editor are unaware of the surrounding content/assets. To reference assets such as sounds, textures, models, or animations. For example you can create a 'UeSoundAsset' schema with three properties: *Id* (required), *Path*, and *Name*. Prepare a `FJsonObject` listing of your assets (see Unreal Engine `AssetRegistry` module documentation) in following format:

```
{
    "UeSoundAsset": [{
            "Id": "_Content_Sounds_MySound",
            "Path": "/Content/Sounds/MySound",
            "Name": "MySound"
        }
        /* other assets */
    ],
    /* other document collections to import */
}
```

Then, import this list into your game data file using the `FCharonCli::Import` method with `EImportMode::Replace` import mode. It's crucial that the *Id* field of imported records remains stable and unchanged across imports for the same assets.

To streamline the process of importing asset paths, consider leveraging the `ICharonEditorModule::OnGameDataPreSynchronization` event. This allows for automatic execution of the import routine each time the **Import** button is clicked in the UI.

After you've imported the asset list into the game data file, you can reference them from your documents by adding a `Document Reference` property with **Reference Type** → **UeSoundAsset** to the schema.

Feedback

We welcome and encourage feedback, particularly bug reports and suggestions, to help improve our tool. If you have any questions or would like to share your thoughts, please join our Discord community or reach out to us via email at support@gamedevware.com.

See also

- Basic Navigation and User Interface Overview

- Creating Document Type (Schema)
- Filling Documents
- Frequently Asked Questions (FAQ)
- Glossary

How to Create Game Data File

To create a new game data file within the Unreal Engine Editor, open the **Content Drawer**, right-click in the desired folder, and select in the **Create Advanced Assets** section **Miscellaneous** → **Game Data** menu option. Name your game data file and proceed according to the instructions in the dialog window that appears.

Step By Step

1. **Open Content Drawer:** Open the *Content Drawer* window in the Unreal Engine Editor.
2. **Select Folder:** Right-click in the desired folder where you want to create the game data file.
3. **Create Game Data:** Navigate to **Create Advanced Assets** → **Miscellaneous** → **Game Data** from the context menu.
4. **Name the File:** In the *Content Drawer* window that appears, enter a name for your game data file.
5. **Check for Errors:** Ensure there are no error messages in the dialog window that opens, then press *Next*.
6. **Wait for Module Generation:** Allow time for the new module to be generated, watching the wizard in the dialog proceed to the next step automatically.
7. **Review Summary:** Check the summary and verify there are no suspicious errors in the *Output Log* window.
8. **Recompile C++ Code:** Use your IDE of choice to recompile the C++ code. Restart Unreal Engine Editor if needed.
9. **Import Game Data:** Reopen the *Content Drawer* window and click the **Import** button.
10. **Select .gdjs File:** Locate and select the *.gdjs* game data file you named in step 4, then click *Ok*.
11. **Choose Game Data Class:** Select the *Game Data* class, which should match the game data file name. If it's not listed, return to step 7.
12. **Save .uasset File:** Save the newly created *.uasset* file after completing the import process.

Throubleshooting

Game data creation or code generation/compilation may encounter issues under certain circumstances:

Insufficient File System Rights or File Creation Errors - **Problem**: Lack of sufficient rights to the OS file system, or errors during file creation (e.g., file name too long, antivirus block). - **Solution**: Check the *Output Log* window for errors or the most recent log file in `<project-dir>\Intermediate\Charon\logs` and attempt to resolve them.

Class Name Collision Within Project - **Solution 1** (Game Data Class Name Collision): Delete the newly created **.gdjs** game data file and the generated module. Then, start over with a new name and clean your `.Target.cs` files from the generated module name. - **Solution 2** (Schema Class Name Collision): Open the game data in another editor (Online, Standalone), rename the schema, and try again.

No Game Data Class in Import Window - **Problem**: The generated game data module is not being compiled. - **Solution**: Ensure it's added to your `<project-name>.Target.cs` and `<project-name>Editor.Target.cs` files as an extra module. If missing, include following expression in both target files:

```
ExtraModuleNames.Add("<module-name>");
```

Additionally, verify that your **.uproject** file includes the generated module definition. If it's absent, add the following module definition to the **Modules** list:

Overview

```json
{
    "Name": "<module-name>",
    "Type": "Runtime",
    "LoadingPhase": "Default"
}
```

See also

- Basic Navigation and User Interface Overview

- Creating Document Type (Schema)

- Filling Documents

- Frequently Asked Questions (FAQ)

- Glossary

Unity Plugin Overview

Charon is a versatile plugin tailored for Unity, designed to facilitate data-driven game design by allowing both developers and game designers to efficiently manage static game data, like units, items, missions, quests, and other. Charon elevates the experience by offering an editing UI directly in your web browser, enabling the modeling of diverse data structures suited to any game genre. It provides a user-friendly interface that requires no special skills for game designers, simplifying the process of data manipulation. For programmers, Charon streamlines development workflows by generating code to load game data seamlessly into the game.

Key Features

- **Data Modeling**: Define game entities like characters, items, missions, quests, and dialogs to meet the specific needs of your game. Interconnect and fill these tables within one UI.

- **Error Control**: Implements validation checks to verify the accuracy of input data, reducing the likelihood of errors that could impact gameplay or development.

- **Code Generation**: Automates the creation of boilerplate code needed to work with your game data, significantly speeding up development time and reducing manual coding errors.

- **Spreadsheet Export/Import**: Offers seamless integration with spreadsheet software, enabling you to effortlessly populate, edit, and manage your game data in a familiar environment.

- **Localization Export/Import**: Simplifies the process of preparing game data for translation, making it straightforward to adapt your game for global audiences.

- **Modding Support**: Empowers your gaming community by providing them with the tools to create and share mods, enhancing the longevity and depth of your game.

- **Dynamic Load**: Facilitates the dynamic loading of game data, enabling features like A/B testing or the ability to push hot updates directly to your players.

Getting Started

Prerequisites

Unity plugin uses `dotnet charon` tool, which is a .NET Core application built for .NET 8.

Windows

1. Download and install NET 8+.
2. Make sure you have write access to `%APPDATA%/Charon`.

MacOS

1. Download and install NET 8+.
2. Make sure you have write access to `~/Library/Application Support/Charon`.
3. Make sure `dotnet` is available from `$PATH`.

Linux

1. Download and install NET 8+.
2. Make sure you have write access to `~/.config`.
3. Make sure `dotnet` is available from `$PATH`.

Checking Available .NET Versions

In terminal window run `dotnet --list-sdks` command:

```
# check for dotnet already installed
dotnet --list-sdks
```

Installation from OpenUPM (recommended)

1. Install the required software for your operating system.
2. Ensure your Unity version is 2021.3 or later.
3. Open the OpenUPM page for the plugin.
4. Click the **Manual Installation** button in the upper right corner and follow the instructions.

Installation from Unity Asset Store

1. Install the required software for your operating system.
2. Ensure your Unity version is 2021.3 or later.
3. Open the Charon plugin in the Unity Asset Store.
4. Click **Add To My Assets**.
5. Open the Unity Package Manager by navigating to **Window → Package Manager**.
6. Wait for the package manager to populate the list.
7. Select **My Assets** from the dropdown in the top left corner.
8. Select **Charon** from the list and click **Download**. If it's already downloaded, you will see an **Import** option.

Installation from GitHub

1. Install the required software for your operating system.
2. Clone or download the plugin source code from the GitHub repository.
3. Create a `<project-dir>/Packages/com.gamedevware.charon` directory.

4. Copy the plugin files from `src/GameDevWare.Charon.Unity/Packages/com.gamedevware.charon` into this directory.

5. Restart Unity if necessary.

Core Concepts

Data-Driven Design Principles

Data-driven design emphasizes controlling gameplay through data rather than source code or blueprints. Game mechanics and processes are determined by structured data files. For example, instead of embedding damage calculations directly in the game's source code, these are defined by data specifying weapon effects and the rules for their application. Similarly, mission progression is not hardcoded; it is outlined in editable text files, making these aspects of game design highly flexible. This approach not only facilitates quick adjustments during development but also simplifies adding modding support post-release.

- Modify Everything! Data-Driven Dynamic Gameplay Effects in 'For Honor' (Video)
- Data-driven Design in Unreal (Article)

Understanding the Plugin's Architecture

Plugin Assets
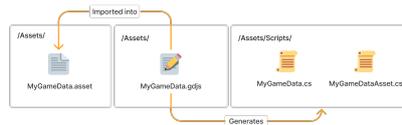
All game data information is stored in a JSON file within your project. The generated source code is used to load this data into the game. Additionally, a `ScriptableObject` asset will be created, which can be used to access game data from your scenes.
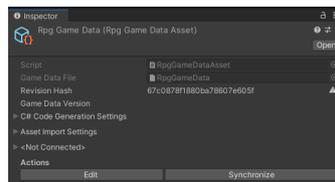
Whenever there is a modification in the data structure within a JSON file, it is necessary to regenerate the C# source code and reimport the *.asset* file. To do this, select the *.asset* file and press the **Synchronize** button.

Working with the Plugin

Creating Game Data

To create a new game data file within the Unity Editor, open the **Project** window, right-click in the desired folder, and select the **Create → Game Data** menu option.

1. Open the **Project** window and navigate to the desired folder.

2. Right-click in the **Project** window and select **Create → Game Data**.

3. Name your game data file and click the **Create** button.

4. Wait for the source code and assets to be created in the specified folder and for the editor to recompile the scripts.

5. Double-click the created *.asset* or *.gdjs* file to start editing.

Editing Game Data

To edit a game data file in the Unity Editor, open the **Project** window, find the corresponding *.gdjs*, *.gdmp*, or *.asset* file, and double-click it. This action opens a new web browser window featuring a user interface for editing the game data. Remember to **Synchronize** assets from the Inspector window after completing your edits.
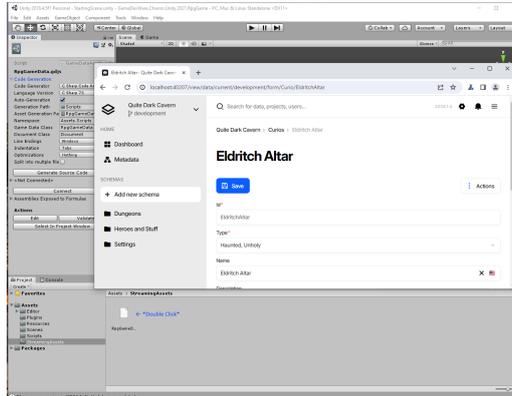
Advanced Features

Localization and Multi-Language Support

Charon facilitates multi-language text support through the `Localizable Text` data type. When creating a *Schema*, properties can be defined with various data types, including `Localizable Text`. Initially, all localizable text defaults to `EN-us` (US English). Additional languages can be added via **Project Settings** → **Internationalization** → **Translation Languages** in the Charon UI.

Exporting/importing localizable data.

Referencing Game Data in Scenes

The Charon plugin introduces a specific type for referencing documents within scenes, named `GameDataDocumentReference`. This type is part of the Charon package. To create such a reference, add a field with the `GameDataDocumentReference` type to your component class.

```
public class HeroComponent : MonoBehaviour
{
  public GameDataDocumentReference heroReference;
}
```

You can then configure it in the Inspector. Here is an example of a **Game Data Document Reference** used to point to a *Hero* document:
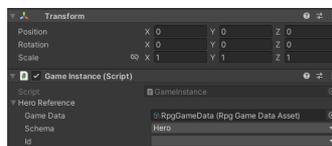
To get an instance of a document in your game code, call the `GameDataDocumentReference.GetReferencedDocument<Hero>()` method.

```
private void OnEnable()
{
  var hero = this.heroReference.GetReferencedDocument<Hero>();
  Debug.Log(hero.Name);
}
```

Work & Build Automation

To facilitate automation of work or builds, a programmatic interface for working with game data is provided. You can read more about it on the CharonCli class documentation page.

Feedback

We welcome and encourage feedback, particularly bug reports and suggestions, to help improve our tool. If you have any questions or would like to share your thoughts, please join our Discord community or reach out to us via email at support@gamedevware.com.
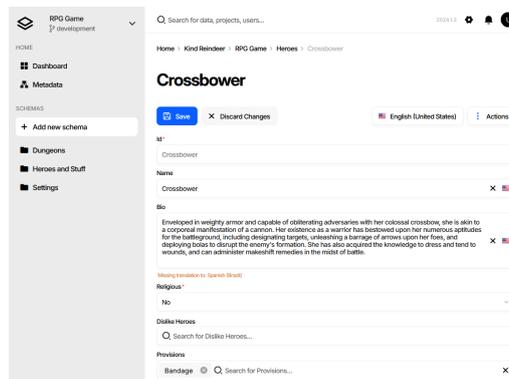
See also

- Basic Navigation and User Interface Overview
- Creating Document Type (Schema)
- Filling Documents
- Frequently Asked Questions (FAQ)
- Glossary

CharonCli Overview

The CharonCli class provides a convenient interface for running `dotnet charon` command-line operations. It simplifies interactions with the Charon tool, enabling developers to manage game data, automate workflows, and integrate with Unity projects. Below is an overview of its methods grouped by purpose.

Game Data Management

- **InitGameDataAsync**: Initializes a GameData file.
- **CreateDocumentAsync**: Creates a document in the specified GameData URL.
- **UpdateDocumentAsync**: Updates a document in the specified GameData URL.
- **DeleteDocumentAsync**: Deletes a document in the specified GameData URL (by document or ID).
- **FindDocumentAsync**: Finds a document in the specified GameData URL by ID.
- **ListDocumentsAsync**: Lists documents in the specified GameData URL with optional filters and sorting.

Import and Export

- **ImportAsync**: Imports documents grouped by schema into a specified GameData URL.
- **ImportFromFileAsync**: Imports documents from a file into a specified GameData URL.
- **ExportAsync**: Exports documents from a GameData URL.
- **ExportToFileAsync**: Exports documents from a GameData URL to a file.

Localization (I18N)

- **I18NImportAsync**: Imports translated documents grouped by schema into a specified GameData URL.
- **I18NImportFromFileAsync**: Imports translated documents from a file into a specified GameData URL.
- **I18NExportAsync**: Exports documents for localization from a GameData URL.
- **I18NExportToFileAsync**: Exports documents for localization from a GameData URL to a file.
- **I18NAddLanguageAsync**: Adds translation languages to a GameData URL.

Patching and Backup

- **CreatePatchAsync**: Compares documents in two GameData URLs and creates a patch representing the difference.
- **CreatePatchToFileAsync**: Compares documents in two GameData URLs and saves the patch to a file.

- **ApplyPatchAsync**: Applies a patch to a specified GameData URL.

- **ApplyPatchFromFileAsync**: Applies a patch from a file to a specified GameData URL.

- **BackupAsync**: Backs up game data with all documents and metadata.

- **BackupToFileAsync**: Backs up game data to a file with all documents and metadata.

- **RestoreAsync**: Restores game data from a backup.

- **RestoreFromFileAsync**: Restores game data from a backup file.

Validation and Code Generation

- **ValidateAsync**: Validates all documents in a GameData URL and returns a report with issues.

- **GenerateCSharpCodeAsync**: Generates C# source code for loading game data into a game's runtime.

- **DumpTemplatesAsync**: Dumps T4 code generation templates into a specified directory.

Tool Utilities

- **GetVersionAsync**: Gets the version number of the Charon tool executable.

- **GetGameDataToolVersionAsync**: Gets the version of the Charon tool used to create a GameData URL.

- **RunCharonAsync**: Runs a specified command with the Charon tool.

- **RunT4Async**: Processes T4 templates using the `dotnet-t4` command-line tool.

See also

- Unity Plugin Overview

- CharonCli class

- Examples of CharonCli class

Migration from Legacy Version (Before 2025.1.*)

> **Warning**
>
> **Before proceeding with the migration, ensure your project is under a source control system (e.g., Git) or that a full backup of your project has been created.** Migration involves modifying and deleting files, and having a backup or version control ensures you can recover in case of unexpected issues.

Install the package with the new version of the plugin via the Unity Asset Store or using OpenUPM (recommended). After installing plugin package you have two options:

Automated Migration

A window will appear offering to perform the migration automatically.

1. Click the **Migrate** button and wait for the process to complete.

2. Once the migration is finished, close the window if everything is successful.

3. If an error occurs, check the **Console** window for details and consider using the *Manual Migration* approach.

Manual Migration

To migrate manually, you will need to remove the old plugin, convert, and configure the game data files:

1. Navigate to the `Assets/Editor/GameDevWare.Charon` folder and delete it.

2. Temporarily move all **.gdjs** and **.gdmp** files from `Assets/StreamingAssets/` to `Assets/`.

3. Select each **.gdjs** or **.gdmp** file and click the **Reimport** button in the **Inspector** window.

4. Replace the old **.asset** file with the newly generated one. If the file did not exist previously, place it anywhere within the boundaries of the **.asmdef** file.

5. Replace the old source code files (**.cs**) with the newly generated ones.

> ### Warning
>
> Preserve the original **.meta** files for **.cs** and **.asset** assets to maintain Unity resource associations and links.

See also

- Unity Plugin Overview

Migrating to Web Application

To migrate to the https://charon.live, you can do it through a *backup <../web/migrating_to_web>* or through the "Connection" mechanism.

In short: you need to create an empty project in at https://charon.live, in Unity Editor in Inspector window click **Connect**, and specify that you want to upload data to the https://charon.live.

Migration with Connection

Be sure to back up your local data before making any connections.

1. At https://charon.live: on the Home screen, click on **Create Project**.

2. Specify the project name, tags, and script language.

3. Click the **Create** button.

4. In the Unity Editor: select your game data **.asset** file in the **Project** window.

5. In the **Inspector** window, expand **<Not Connected>** foldout, click **Connect** button.

6. In the dialog that opens, click on the **Profile → API Keys** link.

7. At https://charon.live: a page of your profile on the *API Keys* management page should have opened in your browser.

8. Click **Create API Key…** button.

9. Fill in the name and expiration time, then click the **Create** button.

10. Click the **Copy** button in the list of keys next to the newly created key labeled "New!".

11. In the Unity Editor: paste the *API Key* into the corresponding field in the **Connect Game Data** window.

12. Check the *Upload local data…* checkbox, it is only available when the selected *Project* is empty and does not contain any data.

13. Click the **Upload** button".

14. Close **Connect Game Data** window

See also

- **Basics**
- Charon Website

Standalone Application Overview

The standalone version of the game development tool is a desktop application that can be installed on a computer, and it allows the user to design and model game data, as well as generate source code for it. The standalone version is typically used by individual game developers or small development teams who want to work offline.
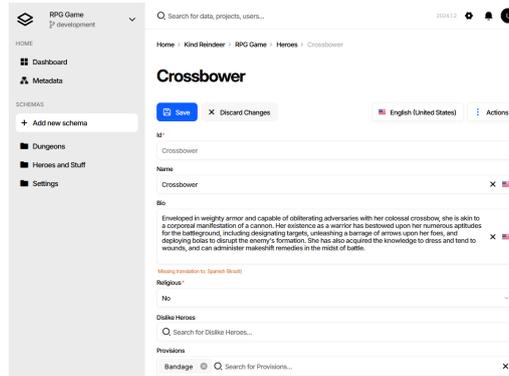
Prerequisites

Standalone application uses *dotnet charon* tool, which is a .NET Core application built for .NET 8.

Windows

1. Download and install NET 8+.
2. Make sure you have write access to `%APPDATA%/Charon` (C:Users%USERNAME%AppDataRoamingCharon).

MacOS

1. Download and install NET 8+.
2. Make sure you have write access to `~/Library/Application Support/Charon`.
3. Make sure `dotnet` is available from `$PATH`.

Linux

1. Download and install NET 8+.
2. Make sure you have write access to `~/.config/Charon`.
3. Make sure `dotnet` is available from `$PATH`.

Checking Available .NET Versions

```
# check for mono already installed
dotnet --list-sdks
```

Installation and Updates

You can use just two commands to install the command line tool, or use a bootstrap script that will check dependencies and installed software, and then download and run the tool for you.

```
# install charon globally (run it once)
dotnet tool install -g dotnet-charon
```

```
# update global tool
dotnet tool update -g dotnet-charon
```

Overview

```
# run tool
dotnet charon INIT ./gamedata.json
```

Two bootstrap scripts which download and run latest version of Charon on your PC:

- `RunCharon.bat` for Windows
- `RunCharon.sh` for Linux or MacOS

Both scripts require the [dotnet](#) tool to be available in `PATH`.

1. **Download one of the scripts into a local folder** `charon`.

   a. [RunCharon.bat (Windows)](#)

   b. [RunCharon.sh (Linux, MacOS)](#)

2. Navigate to the local folder `cd charon`.

3. Run `RunCharon.bat` or `RunCharon.sh` depending on your OS.

4. Wait for the script to automatically download and upgrade `dotnet-charon tool`, and display help text.

5. Create an empty file named `RunCharon.bat INIT gamedata.json`

6. Run in standalone mode: `RunCharon.bat gamedata.json`

Or use following bootstrap script:

Windows

```
rem ##### Load and run bootstrap script #####

@echo off
mkdir Charon
cd Charon
curl -O https://raw.githubusercontent.com/gamedevware/charon/main/scripts/bootstrap/RunCharo
./RunCharon.bat INIT ./gamedata.json

rem ##### Start editor #####

./RunCharon.bat ./gamedata.json --log out
```

Linux, MacOS

```
##### Load and run bootstrap script #####

mkdir Charon
cd Charon
curl -O https://raw.githubusercontent.com/gamedevware/charon/main/scripts/bootstrap/RunCharo
chmod +x RunCharon.sh
        ./RunCharon.sh INIT ./gamedata.json

##### Start editor #####

./RunCharon.sh ./gamedata.json --log out
```

Creating and Editing Game Data

Any empty **gamedata.json** file could be used as starting point for standalone application launch. The editor will automatically fill the empty file with the initial data.

Windows

```
./RunCharon.bat ./gamedata.json --log out
```

Linux, MacOS

```
./RunCharon.sh ./gamedata.json --log out
```

After finishing your work, you could just terminate the process with `CTRL+C` keyboard shortcut or close terminal window.

See also

- [Nuget Package](#)
- [Bootstrap Scripts](#)
- Basic Navigation and User Interface Overview
- Creating Document Type (Schema)
- Filling Documents
- Publication of Game Data
- Generating Source Code
- Frequently Asked Questions (FAQ)
- Glossary

Web Application Overview

The web version of the Charon provides a collaborative work environment where game designers can work together to create engaging gameplay experiences. The core concepts of collaborative work include workspaces and projects. A workspace is a virtual location where projects are located. The subscription and all limitations are bound to the workspace, meaning that all projects within the workspace are subject to the same subscription and limitations.

A project is a virtual location for storing game data, localization settings, backups, branches, and members. When a project is created,

the user becomes its owner and can invite other members to join.
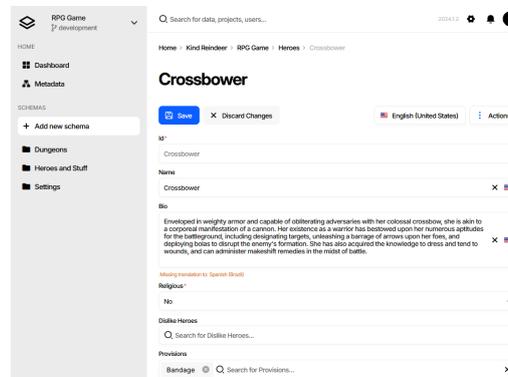
Starting with a new Project

1. Visit the charon.live website and click on the *Register* button to create a new account.

2. Fill out the registration form with your desired username and password, and click on the "Create" button to create your account.

3. After successfully logging in, you will be directed to the workspace page.

4. If you're a new user, the workspace page will be empty, with no projects listed. Click on the *Create project* button to create your first project.

5. On the "Create Project" page, fill in the name of your project and any other basic information you want to include, and click on the "Create" button to create your project.

6. After creating the project, you'll be redirected to the project's dashboard page, which provides an overview of the project and allows you to start modelling game data.

See also

- [Charon Website](#)
- CLI Access to Web Project
- Migrating to Web Application
- Basic Navigation and User Interface Overview
- Creating Document Type (Schema)
- Filling Documents
- Publication of Game Data
- Generating Source Code
- Working with Source Code (C# 4.0)
- Working with Source Code (C# 7.3)
- Working with Source Code (TypeScript)
- Frequently Asked Questions (FAQ)
- Glossary

CLI Access to charon.live

The web version of the Charon provides a REST API and CLI for accessing and modifying game data. To access the API, users need to generate an `API Key` in the *API Keys* section of their *User Profile.*

With the API Key, Charon can easily be integrated into existing game development workflows. For example, the API Key can be used to export game data from the web into a local GIT repository

Step By Step

To generate an API Key:

1. Navigate to the *API Keys* section and click on the *Generate API Key…* button.

2. Copy the generated `API Key`.

3. Use the `API Key` in *the `Authenticate`* header to access the REST API or in the `--credentials` parameter of the CLI.

See also

- Command Line Interface (CLI)
- Migrating to Web
- REST API
- DATA EXPORT Command
- DATA IMPORT Command
- GENERATE CSHARPCODE Command

Migrating to Web Application

To migrate to the web application, you only need to backup your game data in your current editor and restore it in the web application.

In short: project settings include a backup and restore feature. You make a backup in one place, and you can restore it in another. This is how you can transfer game data to another project.

Backup Data Step by Step

1. Open your game data in the editor (Standalone, Unity …)
2. In the bottom left corner, click on the gear icon "Settings"
3. In the left menu, select "Backup"
4. On the backup management page, click the "Backup" button
5. Choose "File" as the destination to save the backup and click "Next"
6. On the "Format" step, select "JSON" and click "Backup"
7. On the "Summary" step, click the link with a file name like `backup_2023_11_10_11_27.json` and save it to your computer.

Restoring Backup in the Web Application

1. On the Home screen, click on "Create Project"
2. Specify the project name, tags, and script language
3. Click the "Create" button
4. In the left menu, select "Backup"
5. On the backup management page, click the "Restore" button
6. Choose "File" as the source of the restore and click "Next"
7. Select the input file, the one you have after the backup, such as `backup_2023_11_10_11_27.json`, and click "Restore"
8. Done

Any data that was in this project at the time before the restore will be lost.

See also

- Basic Navigation and User Interface Overview
- Publication of Game Data
- Generating Source Code
- Frequently Asked Questions (FAQ)
- Glossary

Roles and Permissions

The web version of the Charon provides a system of roles and permissions to manage access and control over your game development projects. Each role is designed to provide specific levels of access and functionality within the platform. Here are the key roles , along with their respective permissions:

Viewer Role:

- View Documents: Users with the Viewer role can access and view documents within projects.
- Export Data: Viewers can export data from the platform.
- Access Project Settings: They can access and view project settings.

Editor Role:

- View Documents: Editors can view documents.
- Edit Documents: Editors have the ability to edit documents within projects.
- Import Data: They can import data into the platform.

- Access Project Settings: Similar to Viewers, Editors can access and view project settings.

Designer Role:

- Change Document Structure: Designers have the privilege to modify the structure of documents.

- View Documents: Designers can view documents.

- Edit Documents: They can edit documents.

- Import Data: Designers can import data.

- Access Project Settings: They can access and view project settings.

Administrator Role:

- Make and Restore Backups: Administrators have the authority to create and restore backups of project data.

- Grant or Revoke Permissions: They can grant or revoke permissions for users within the project.

- Change Project Settings: Administrators can modify various project settings to tailor the environment to their needs.

- View Documents: Administrators can view documents.

- Edit Documents: They can edit documents.

- Import Data: Administrators can import data.

Workspace Administrator and Workspace Owner Role:

- They have the same permissions as Administrators, and they also have the ability to delete and transfer projects.

See also

- Overview
- Web-based Application

REST API

The web version of the Charon provides a experimental REST API feature.

Testing REST API

You can utilize the Swagger UI to perform test requests. In the *Swagger UI*, click on the *Authorize* button and paste your API Key for authentication.

Working with REST API

To make requests, you will need an API Key obtained from your profile page. Add the Authorization: `Basic <api-key>` header to all of your HTTP requests. Also is recommended to provide correct User-Agent header.

Authentication

POST /token
   Authenticate with oAuth protocol.

   Status Codes:
- 200 OK – Authorization granted result.

- 400 Bad Request – Authorization failed result.

POST /auth/one-time-code/

Overview

Request one-time code for current authentication credential. It could be used as 'x-authorization' query parameter instead of 'Authorization' header for WebSockets and download requests.

Status Codes:
- 200 OK – One-time code. It has short expiration time.
- default – Operation failure response.

Request Headers:
- **Authorization** – Authorization header. (Required)

POST /auth/flow/password/

Begin authentication flow with password request.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST /auth/flow/email-code/

Continue authentication flow with TOTP code from email.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST /auth/flow/api-key/

Start authentication flow with API Key.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST /auth/flow/on-behalf/

Start authentication flow on behalf of another user (Administrator only).

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST /auth/flow/oauth2/{authenticationProvider}/prepare/

Prepare OAuth2 sign-in URL.

Parameters:
- **authenticationProvider** (*string*) – Type of OAuth2 provider.

Status Codes:
- 200 OK – Sign-in URL location.
- default – Operation failure response.

POST /auth/flow/oauth2/{authenticationProvider}/complete/

Complete OAuth2 authentication.

Parameters:
- **authenticationProvider** (*string*) – Type of OAuth2 provider.

Query Parameters:
- **code** (*string*) – The authorization code received from the authorization server.
- **state** (*string*) – An opaque value used by the client to maintain state between the request and callback. (Required)
- **error** (*string*) – A single error code.
- **error_description** (*string*) – Human-readable text providing additional information.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST /token

Authenticate with oAuth protocol.

Overview

Status Codes:
- 200 OK – Authorization granted result.
- 400 Bad Request – Authorization failed result.

POST `/auth/one-time-code/`

Request one-time code for current authentication credential. It could be used as 'x-authorization' query parameter instead of 'Authorization' header for WebSockets and download requests.

Status Codes:
- 200 OK – One-time code. It has short expiration time.
- default – Operation failure response.

Request Headers:
- **Authorization** – Authorization header. (Required)

POST `/auth/flow/password/`

Begin authentication flow with password request.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST `/auth/flow/email-code/`

Continue authentication flow with TOTP code from email.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST `/auth/flow/api-key/`

Start authentication flow with API Key.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST `/auth/flow/on-behalf/`

Start authentication flow on behalf of another user (Administrator only).

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

POST `/auth/flow/oauth2/{authenticationProvider}/prepare/`

Prepare OAuth2 sign-in URL.

Parameters:
- **authenticationProvider** (*string*) – Type of OAuth2 provider.

Status Codes:
- 200 OK – Sign-in URL location.
- default – Operation failure response.

POST `/auth/flow/oauth2/{authenticationProvider}/complete/`

Complete OAuth2 authentication.

Parameters:
- **authenticationProvider** (*string*) – Type of OAuth2 provider.

Query Parameters:
- **code** (*string*) – The authorization code received from the authorization server.
- **state** (*string*) – An opaque value used by the client to maintain state between the request and callback. (Required)
- **error** (*string*) – A single error code.
- **error_description** (*string*) – Human-readable text providing additional information.

Status Codes:
- 200 OK – Authentication flow stage.
- default – Operation failure response.

Overview

DataSource

PUT /datasource/{dataSourceId}/transaction/
Wait for data source availability and begin new transaction. Identifier specified in request later could be used with other request in `transaction` parameter.

> **Parameters:**
> - **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
>
> **Status Codes:**
> - 204 No Content – The transaction has started.
> - default – Operation failure response.

POST /datasource/{dataSourceId}/transaction/{transactionId}/
Commit pending transaction.

> **Parameters:**
> - **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
> - **transactionId** (*string*) – Id or name of transaction. Optional.
>
> **Status Codes:**
> - 204 No Content – The transaction has been successfully completed.
> - default – Operation failure response.

DELETE /datasource/{dataSourceId}/transaction/{transactionId}/
Reject pending transaction.

> **Parameters:**
> - **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
> - **transactionId** (*string*) – Id or name of transaction. Optional.
>
> **Status Codes:**
> - 204 No Content – The transaction has either been aborted or has already failed.
> - default – Operation failure response.

GET /datasource/{dataSourceId}/collection/{schemaNameOrId}/
Find document by it's id or unique property value.

> **Parameters:**
> - **schemaNameOrId** (*string*) – Id or name of schema.
> - **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
>
> **Query Parameters:**
> - **uniqueSchemaPropertyNameOrId** (*string*) – Document unique property's name. For example 'Id'. (Required)
> - **uniqueSchemaPropertyValue** (*string*) – Document's unique property value. For example it's 'Id' property. (Required)
> - **transactionId** (*string*) – Id or name of transaction.
>
> **Status Codes:**
> - 200 OK – Found document or null.
> - default – Operation failure response.

PUT /datasource/{dataSourceId}/collection/{schemaNameOrId}/
Create document.

> **Parameters:**
> - **schemaNameOrId** (*string*) – Id or name of schema.
> - **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
>
> **Query Parameters:**
> - **validationOptions** (*array*) – Data source validation options. (Required)
> - **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Created document.
- default – Operation failure response.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/
   Update document.

Parameters:
- **schemaNameOrId** (*string*) – Id or name of schema.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **validationOptions** (*array*) – Data source validation options. (Required)
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Updated document.
- default – Operation failure response.

DELETE /datasource/{dataSourceId}/collection/{schemaNameOrId}/
   Delete document by it's id.

Parameters:
- **schemaNameOrId** (*string*) – Id or name of schema.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **id** (*string*) – Document's id. (Required)
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Deleted document or null.
- default – Operation failure response.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/documents/
   List documents.

Parameters:
- **schemaNameOrId** (*string*) – Id or name of schema.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Found documents.
- default – Operation failure response.

PUT /datasource/{dataSourceId}/collection/{schemaNameOrId}/documents/
   Bulk change documents.

Parameters:
- **schemaNameOrId** (*string*) – Id or name of schema.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **validationOptions** (*array*) – Data source validation options. (Required)
- **dryRun** (*boolean*) – Perform dry run of operation and don't persist changes'. (Required)
- **importMode** (*string*) – Import mode. (Required)
- **transactionId** (*string*) – Id or name of transaction.

23

Status Codes:
- 200 OK – Created document.
- default – Operation failure response.

GET /datasource/{dataSourceId}/documents/query/
Query documents from all collections.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **query** (*string*) – Keyword to lookup in documents. (Required)
- **limit** (*integer*) – Maximum number of documents to return.
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Found document stream where each event is instance of 'ListResult'. Empty results are ommited.

POST /datasource/{dataSourceId}/documents/query/
Pick multiple documents by their unique properties e.g. batched find request. Max documents per request is - 20.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Pick results in same order as requested.
- default – Operation failure response.

POST /datasource/{dataSourceId}/converter/raw/
Convert specified game data documents from request body to JSON format and return it without response wrapper.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – Converted game data document.
- default – Operation failure response.

GET /datasource/{dataSourceId}/collections/raw/
Export documents from multiple collections into downloadable format without response wrapper.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **transactionId** (*string*) – Id or name of transaction.
- **exportMode** (*string*) – Export mode. (Required)
- **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.
- **properties** (*array*) – List of properties on schemas to export. Id property is always exported. Empty list mean all properties.
- **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:
- 200 OK – Exported documents.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/raw/`

Backup data source into downloadable format without response wrapper.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:
- 200 OK – Exported documents.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/loading-progress/`

Get data source's loading progress.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – Data source's capabilities and loading progress.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/stats/`

Get data source's statistics.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Data source's statistics.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/collections/`

Export documents from multiple collections.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **exportMode** (*string*) – Export mode. (Required)
- **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.
- **properties** (*array*) – List of properties on schemas to export. Id property is always exported. Empty list mean all properties.
- **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Exported documents.
- default – Operation failure response.

`PUT /datasource/{dataSourceId}/collections/`

Import documents into multiple collections.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:	• **importMode** (*string*) – Import mode. (Required)
	• **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.
	• **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
	• **validationOptions** (*array*) – Data source validation options. (Required)
	• **dryRun** (*boolean*) – Perform dry run of operation and don't persist changes'. (Required)
	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Import report.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/
Backup data source.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Status Codes:	• 200 OK – Backed up data source.
	• default – Operation failure response.

PUT /datasource/{dataSourceId}/
Restore data source from specified documents.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Status Codes:	• 200 OK – Data source restoration result.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/validity/
Validadate data source with specified requirements/parameters.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **transactionId** (*string*) – Id or name of transaction.
	• **validationOptions** (*array*) – Data source validation options. (Required)
Status Codes:	• 200 OK – Data source's configuration.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/source-code/
Generate source code for data source.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.
Status Codes:	• 200 OK – Source code packed into .zip archive.

GET /datasource/{dataSourceId}/source-code/templates/
Get T4 templates for generating source code.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:
- 200 OK – Source code generation templates packed into .zip archive.

`PUT /datasource/{dataSourceId}/transaction/`

Wait for data source availability and begin new transaction. Identifier specified in request later could be used with other request in `transaction` parameter.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 204 No Content – The transaction has started.
- default – Operation failure response.

`POST /datasource/{dataSourceId}/transaction/{transactionId}/`

Commit pending transaction.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
- **transactionId** (*string*) – Id or name of transaction. Optional.

Status Codes:
- 204 No Content – The transaction has been successfully completed.
- default – Operation failure response.

`DELETE /datasource/{dataSourceId}/transaction/{transactionId}/`

Reject pending transaction.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
- **transactionId** (*string*) – Id or name of transaction. Optional.

Status Codes:
- 204 No Content – The transaction has either been aborted or has already failed.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/collection/{schemaNameOrId}/`

Find document by it's id or unique property value.

Parameters:
- **schemaNameOrId** (*string*) – Id or name of schema.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **uniqueSchemaPropertyNameOrId** (*string*) – Document unique property's name. For example 'Id'. (Required)
- **uniqueSchemaPropertyValue** (*string*) – Document's unique property value. For example it's 'Id' property. (Required)
- **transactionId** (*string*) – Id or name of transaction.

Status Codes:
- 200 OK – Found document or null.
- default – Operation failure response.

`PUT /datasource/{dataSourceId}/collection/{schemaNameOrId}/`

Create document.

Parameters:
- **schemaNameOrId** (*string*) – Id or name of schema.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

27

Query Parameters:	• **validationOptions** (*array*) – Data source validation options. (Required)
	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Created document.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/
  Update document.

Parameters:	• **schemaNameOrId** (*string*) – Id or name of schema.
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **validationOptions** (*array*) – Data source validation options. (Required)
	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Updated document.
	• default – Operation failure response.

DELETE /datasource/{dataSourceId}/collection/{schemaNameOrId}/
  Delete document by it's id.

Parameters:	• **schemaNameOrId** (*string*) – Id or name of schema.
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **id** (*string*) – Document's id. (Required)
	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Deleted document or null.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/documents/
  List documents.

Parameters:	• **schemaNameOrId** (*string*) – Id or name of schema.
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Found documents.
	• default – Operation failure response.

PUT /datasource/{dataSourceId}/collection/{schemaNameOrId}/documents/
  Bulk change documents.

| **Parameters:** | • **schemaNameOrId** (*string*) – Id or name of schema. |
| | • **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project. |

Query Parameters:	• **validationOptions** (*array*) – Data source validation options. (Required)
	• **dryRun** (*boolean*) – Perform dry run of operation and don't persist changes'. (Required)
	• **importMode** (*string*) – Import mode. (Required)
	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Created document.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/documents/query/

Query documents from all collections.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **query** (*string*) – Keyword to lookup in documents. (Required)
	• **limit** (*integer*) – Maximum number of documents to return.
	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Found document stream where each event is instance of 'ListResult'. Empty results are ommited.

POST /datasource/{dataSourceId}/documents/query/

Pick multiple documents by their unique properties e.g. batched find request. Max documents per request is - 20.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	• 200 OK – Pick results in same order as requested.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/converter/raw/

Convert specified game data documents from request body to JSON format and return it without response wrapper.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Status Codes:	• 200 OK – Converted game data document.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/collections/raw/

Export documents from multiple collections into downloadable format without response wrapper.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:	
	• **transactionId** (*string*) – Id or name of transaction.
	• **exportMode** (*string*) – Export mode. (Required)
	• **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.
	• **properties** (*array*) – List of properties on schemas to export. Id property is always exported. Empty list mean all properties.
	• **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
	• **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.
Status Codes:	
	• 200 OK – Exported documents.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/raw/

Backup data source into downloadable format without response wrapper.

Parameters:	
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.
Status Codes:	
	• 200 OK – Exported documents.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/loading-progress/

Get data source's loading progress.

Parameters:	
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Status Codes:	
	• 200 OK – Data source's capabilities and loading progress.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/stats/

Get data source's statistics.

Parameters:	
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **transactionId** (*string*) – Id or name of transaction.
Status Codes:	
	• 200 OK – Data source's statistics.
	• default – Operation failure response.

GET /datasource/{dataSourceId}/collections/

Export documents from multiple collections.

Parameters:	
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **exportMode** (*string*) – Export mode. (Required)

- **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.

- **properties** (*array*) – List of properties on schemas to export. Id property is always exported. Empty list mean all properties.

- **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.

- **transactionId** (*string*) – Id or name of transaction.

Status Codes:

- 200 OK – Exported documents.

- default – Operation failure response.

`PUT /datasource/{dataSourceId}/collections/`
   Import documents into multiple collections.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **importMode** (*string*) – Import mode. (Required)

- **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.

- **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.

- **validationOptions** (*array*) – Data source validation options. (Required)

- **dryRun** (*boolean*) – Perform dry run of operation and don't persist changes'. (Required)

- **transactionId** (*string*) – Id or name of transaction.

Status Codes:

- 200 OK – Import report.

- default – Operation failure response.

`GET /datasource/{dataSourceId}/`
   Backup data source.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:

- 200 OK – Backed up data source.

- default – Operation failure response.

`PUT /datasource/{dataSourceId}/`
   Restore data source from specified documents.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:

- 200 OK – Data source restoration result.

- default – Operation failure response.

`POST /datasource/{dataSourceId}/validity/`
   Validadate data source with specified requirements/parameters.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **transactionId** (*string*) – Id or name of transaction.

- **validationOptions** (*array*) – Data source validation options. (Required)

31

Status Codes:
- 200 OK – Data source's configuration.
- default – Operation failure response.

`POST /datasource/{dataSourceId}/source-code/`
Generate source code for data source.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:
- 200 OK – Source code packed into .zip archive.

`GET /datasource/{dataSourceId}/source-code/templates/`
Get T4 templates for generating source code.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:
- 200 OK – Source code generation templates packed into .zip archive.

DataSourceCapabilities

`GET /datasource/{dataSourceId}/capabilities/`
Get data source's capabilities.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – Data source's capabilities.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/capabilities/`
Get data source's capabilities.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – Data source's capabilities.
- default – Operation failure response.

UserPresence

`GET /datasource/{dataSourceId}/present-users/`
Get list of users present in specificated data source.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – List of user presence in data source.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/present-users/`
Get list of users present in specificated data source.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:

- 200 OK – List of user presence in data source.
- default – Operation failure response.

Processes

`GET /datasource/{dataSourceId}/process/`
  List processes.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.

Status Codes:

- 200 OK – Process state.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/process/{processId}/`
  Get process's state.

Parameters:

- **processId** (*integer*) – Id of process.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:

- 200 OK – Process state.
- default – Operation failure response.

`POST /datasource/{dataSourceId}/process/{processId}/`
  Stop process.

Parameters:

- **processId** (*integer*) – Id of process.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **stopReason** (*string*) – Reason why process has been stopped. (Required)

Status Codes:

- 200 OK – Stopped process state.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/process/{processId}/result/raw/`
  Get process's execution result without response wrapper.

Parameters:

- **processId** (*integer*) – Id of process.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:

- 200 OK – Process result.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/process/`
  List processes.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.

Status Codes:

- 200 OK – Process state.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/process/{processId}/`
Get process's state.

Parameters:

- **processId** (*integer*) – Id of process.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:

- 200 OK – Process state.
- default – Operation failure response.

`POST /datasource/{dataSourceId}/process/{processId}/`
Stop process.

Parameters:

- **processId** (*integer*) – Id of process.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **stopReason** (*string*) – Reason why process has been stopped. (Required)

Status Codes:

- 200 OK – Stopped process state.
- default – Operation failure response.

`GET /datasource/{dataSourceId}/process/{processId}/result/raw/`
Get process's execution result without response wrapper.

Parameters:

- **processId** (*integer*) – Id of process.
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **download** (*boolean*) – Set "Content-Disposition" header in order to make the browser download the result.

Status Codes:

- 200 OK – Process result.
- default – Operation failure response.

Formulas

`GET /datasource/{dataSourceId}/formula/type/`
List formula types.

Parameters:

- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:

- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
- **query** (*string*) – Any value to search in type name.

Status Codes:
- 200 OK – List of types.
- default – Operation failure response.

GET /datasource/{dataSourceId}/formula/type/

List formula types.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
- **query** (*string*) – Any value to search in type name.

Status Codes:
- 200 OK – List of types.
- default – Operation failure response.

AiCompletion

POST /datasource/{dataSourceId}/completion/schema/

Suggest schema structure with specified AI tool.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – AI completion response with schema structure in natural language.
- default – Operation failure response.

POST /datasource/{dataSourceId}/completion/schema/icon/

Suggest an icon for schema using AI tool.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **iconSet** (*string*) – Name of icon set to select from. (Required)
- **schemaName** (*string*) – Name of the schema to generate icon for. (Required)
- **schemaDescription** (*string*) – Description of the schema to generate icon for.

Status Codes:
- 200 OK – AI completion response with schema structure in natural language.
- default – Operation failure response.

POST /datasource/{dataSourceId}/completion/thread/{threadId}/

Send AI chat message to specified chat thread.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
- **threadId** (*string*) – Id of AI chat thread.

Status Codes:
- 200 OK – AI completion response with schema structure in natural language.
- default – Operation failure response.

DELETE /datasource/{dataSourceId}/completion/thread/{threadId}/

Delete existing AI chat thread.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
- **threadId** (*string*) – Id of AI chat thread.

Status Codes:
- 204 No Content – Thread has been deleted.
- default – Operation failure response.

POST /datasource/{dataSourceId}/completion/schema/
  Suggest schema structure with specified AI tool.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Status Codes:
- 200 OK – AI completion response with schema structure in natural language.
- default – Operation failure response.

POST /datasource/{dataSourceId}/completion/schema/icon/
  Suggest an icon for schema using AI tool.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:
- **iconSet** (*string*) – Name of icon set to select from. (Required)
- **schemaName** (*string*) – Name of the schema to generate icon for. (Required)
- **schemaDescription** (*string*) – Description of the schema to generate icon for.

Status Codes:
- 200 OK – AI completion response with schema structure in natural language.
- default – Operation failure response.

POST /datasource/{dataSourceId}/completion/thread/{threadId}/
  Send AI chat message to specified chat thread.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
- **threadId** (*string*) – Id of AI chat thread.

Status Codes:
- 200 OK – AI completion response with schema structure in natural language.
- default – Operation failure response.

DELETE /datasource/{dataSourceId}/completion/thread/{threadId}/
  Delete existing AI chat thread.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
- **threadId** (*string*) – Id of AI chat thread.

Status Codes:
- 204 No Content – Thread has been deleted.
- default – Operation failure response.

MachineTranslation

POST /datasource/{dataSourceId}/translation/
  Machine translate portion of game data. First language in the list is a source language.

Parameters:
- **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.

Query Parameters:	• **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.
	• **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
	• **translationMode** (*string*) – Translation mode. (Required)
Status Codes:	• 202 Accepted – Translation process has been started.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/translation/
    Machine translate specified document. First language in the list is a source language.

Parameters:	• **schemaNameOrId** (*string*) – Id or name of schema.
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
	• **translationMode** (*string*) – Translation mode. (Required)
Status Codes:	• 200 OK – Translated document.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/translation/
    Machine translate portion of game data. First language in the list is a source language.

Parameters:	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **schemas** (*array*) – List of schemas to export/import. Empty list mean all schemas.
	• **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
	• **translationMode** (*string*) – Translation mode. (Required)
Status Codes:	• 202 Accepted – Translation process has been started.
	• default – Operation failure response.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/translation/
    Machine translate specified document. First language in the list is a source language.

Parameters:	• **schemaNameOrId** (*string*) – Id or name of schema.
	• **dataSourceId** (*string*) – Id of data source. Usually it is a branchId from one of branches from Project.
Query Parameters:	• **languages** (*array*) – List of languages on schemas to export. Empty list mean all languages.
	• **translationMode** (*string*) – Translation mode. (Required)
Status Codes:	• 200 OK – Translated document.
	• default – Operation failure response.

Preferences

GET /project/{projectId}/preferences/
    Get project team-shared preferences.

Parameters:	• **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 200 OK – Preferences object.
- default – Operation failure response.

PUT /project/{projectId}/preferences/
  Save project team-shared preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Preferences has been saved.
  - default – Operation failure response.

PATCH /project/{projectId}/preferences/
  Patch project team-shared preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Patch has been applyed.
  - default – Operation failure response.

GET /project/{projectId}/preferences/user/
  Get project user's preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 200 OK – Preferences object.
  - default – Operation failure response.

PUT /project/{projectId}/preferences/user/
  Save project user's preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Preferences has been saved.
  - default – Operation failure response.

PATCH /project/{projectId}/preferences/user/
  Patch project user's preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Patch has been applied.
  - default – Operation failure response.

GET /workspace/{workspaceId}/preferences/
  Get workspace team-shared preferences.

  Parameters:
  - **workspaceId** (*string*) – Id of workspace.

  Status Codes:
  - 200 OK – Preferences object.
  - default – Operation failure response.

PUT /workspace/{workspaceId}/preferences/
  Save workspace team-shared preferences.

  Parameters:
  - **workspaceId** (*string*) – Id of workspace.

  Status Codes:
  - 204 No Content – Preferences has been saved.
  - default – Operation failure response.

PATCH /workspace/{workspaceId}/preferences/

Patch workspace team-shared preferences.

> **Parameters:**
> - **workspaceId** (*string*) – Id of workspace.
>
> **Status Codes:**
> - 204 No Content – Patch has been applyed.
> - default – Operation failure response.

`GET /workspace/{workspaceId}/preferences/user/`
Get user's workspace preferences.

> **Parameters:**
> - **workspaceId** (*string*) – Id of workspace.
>
> **Status Codes:**
> - 200 OK – Preferences object.
> - default – Operation failure response.

`PUT /workspace/{workspaceId}/preferences/user/`
Save user's workspace preferences.

> **Parameters:**
> - **workspaceId** (*string*) – Id of workspace.
>
> **Status Codes:**
> - 204 No Content – Preferences has been saved.
> - default – Operation failure response.

`PATCH /workspace/{workspaceId}/preferences/user/`
Patch user's workspace preferences.

> **Parameters:**
> - **workspaceId** (*string*) – Id of workspace.
>
> **Status Codes:**
> - 204 No Content – Patch has been applyed.
> - default – Operation failure response.

`DELETE /preferences/user/`
Reset all user's preferences.

> **Status Codes:**
> - 204 No Content – Preferences has been reset.
> - default – Operation failure response.

`GET /preferences/`
Get default preferences.

> **Status Codes:**
> - 200 OK – Preferences object.
> - default – Operation failure response.

`PUT /preferences/`
Save default preferences.

> **Status Codes:**
> - 204 No Content – Preferences has been saved.
> - default – Operation failure response.

`PATCH /preferences/`
Patch default preferences.

> **Status Codes:**
> - 204 No Content – Patch has been applyed.
> - default – Operation failure response.

`GET /project/{projectId}/preferences/`
Get project team-shared preferences.

> **Parameters:**
> - **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 200 OK – Preferences object.
- default – Operation failure response.

PUT /project/{projectId}/preferences/
  Save project team-shared preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Preferences has been saved.
  - default – Operation failure response.

PATCH /project/{projectId}/preferences/
  Patch project team-shared preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Patch has been applyed.
  - default – Operation failure response.

GET /project/{projectId}/preferences/user/
  Get project user's preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 200 OK – Preferences object.
  - default – Operation failure response.

PUT /project/{projectId}/preferences/user/
  Save project user's preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Preferences has been saved.
  - default – Operation failure response.

PATCH /project/{projectId}/preferences/user/
  Patch project user's preferences.

  Parameters:
  - **projectId** (*string*) – Id of project. Project are located within workspace.

  Status Codes:
  - 204 No Content – Patch has been applied.
  - default – Operation failure response.

GET /workspace/{workspaceId}/preferences/
  Get workspace team-shared preferences.

  Parameters:
  - **workspaceId** (*string*) – Id of workspace.

  Status Codes:
  - 200 OK – Preferences object.
  - default – Operation failure response.

PUT /workspace/{workspaceId}/preferences/
  Save workspace team-shared preferences.

  Parameters:
  - **workspaceId** (*string*) – Id of workspace.

  Status Codes:
  - 204 No Content – Preferences has been saved.
  - default – Operation failure response.

PATCH /workspace/{workspaceId}/preferences/

Patch workspace team-shared preferences.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 204 No Content – Patch has been applyed.
- default – Operation failure response.

`GET /workspace/{workspaceId}/preferences/user/`
Get user's workspace preferences.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Preferences object.
- default – Operation failure response.

`PUT /workspace/{workspaceId}/preferences/user/`
Save user's workspace preferences.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 204 No Content – Preferences has been saved.
- default – Operation failure response.

`PATCH /workspace/{workspaceId}/preferences/user/`
Patch user's workspace preferences.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 204 No Content – Patch has been applyed.
- default – Operation failure response.

`DELETE /preferences/user/`
Reset all user's preferences.

Status Codes:
- 204 No Content – Preferences has been reset.
- default – Operation failure response.

`GET /preferences/`
Get default preferences.

Status Codes:
- 200 OK – Preferences object.
- default – Operation failure response.

`PUT /preferences/`
Save default preferences.

Status Codes:
- 204 No Content – Preferences has been saved.
- default – Operation failure response.

`PATCH /preferences/`
Patch default preferences.

Status Codes:
- 204 No Content – Patch has been applyed.
- default – Operation failure response.

User

`GET /user/`
Get all available users.

41

Query Parameters:	• **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
Status Codes:	• **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
	• **query** (*string*) – Any value to search in user name or email.
	• 200 OK – List of users.
	• default – Operation failure response.

PUT /user/
  Create user with specified parameters.

    Status Codes:
        • 201 Created – User has been created.

        • default – Operation failure response.

POST /user/public/
  Get public profiles of users by their ids.

    Status Codes:
        • 200 OK – List of user public profiles.

        • default – Operation failure response.

GET /user/me/
  Get current user.

    Status Codes:
        • 200 OK – Found user.

        • default – Operation failure response.

PUT /user/password-reset/
  Request password reset.

    Status Codes:
        • 204 No Content – Password reset request has been accepted.

        • default – Operation failure response.

POST /user/password-reset/
  Change user password by using code from email.

    Status Codes:
        • 204 No Content – Password has been reset.

        • default – Operation failure response.

GET /user/{userId}/
  Get user by id.

    Parameters:
        • **userId** (*string*) – Id of user.
    Status Codes:
        • 200 OK – Found user.

        • default – Operation failure response.

POST /user/{userId}/
  Update user with new parameters.

    Parameters:
        • **userId** (*string*) – Id of user.
    Status Codes:
        • 200 OK – Updated workspace.

        • default – Operation failure response.

DELETE /user/{userId}/
  Strip personal information from user, quit all groups and block any access to this user.

    Parameters:
        • **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – User has been soft-deleted.
- default – Operation failure response.

`GET /user/{userId}/public/`

Get user public profile by id.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Found user.
- default – Operation failure response.

`POST /user/{userId}/login/password/`

Change user password by using temporary code or old password.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – Password has been changed.
- default – Operation failure response.

`POST /user/{userId}/mfa/email-code/`

Configure email-code multi-factor authentication.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – Multi-factor authentication has been configured.
- default – Operation failure response.

`DELETE /user/{userId}/login/tokens/`

Revoke all issues tokens for specified user.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – All tokens have been revokes. It is required to re-authenticate after this call.
- default – Operation failure response.

`POST /user/{userId}/login/api-key/`

Add API key login to user.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – List of workspaces.
- default – Operation failure response.

`DELETE /user/{userId}/login/api-key/`

Delete API key login from user.

Parameters:
- **userId** (*string*) – Id of user.

Query Parameters:
- **id** (*string*) – Id of UserLogin with API key. (Required)

Status Codes:
- 204 No Content – API key has been deleted and no longer valid.
- default – Operation failure response.

`POST /user/{userId}/invitations/{invitationId}/`

Accept invitation.

Parameters:
- **userId** (*string*) – Id of user.
- **invitationId** (*string*) – Id of invitation.

43

Status Codes:
- 204 No Content – Invitation has been accepted.
- default – Operation failure response.

`DELETE /user/{userId}/invitations/{invitationId}/`
Decline invitation.

Parameters:
- **userId** (*string*) – Id of user.
- **invitationId** (*string*) – Id of invitation.

Status Codes:
- 204 No Content – Invitation has been dismissed.
- default – Operation failure response.

`GET /user/`
Get all available users.

Query Parameters:
- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
- **query** (*string*) – Any value to search in user name or email.

Status Codes:
- 200 OK – List of users.
- default – Operation failure response.

`PUT /user/`
Create user with specified parameters.

Status Codes:
- 201 Created – User has been created.
- default – Operation failure response.

`POST /user/public/`
Get public profiles of users by their ids.

Status Codes:
- 200 OK – List of user public profiles.
- default – Operation failure response.

`GET /user/me/`
Get current user.

Status Codes:
- 200 OK – Found user.
- default – Operation failure response.

`PUT /user/password-reset/`
Request password reset.

Status Codes:
- 204 No Content – Password reset request has been accepted.
- default – Operation failure response.

`POST /user/password-reset/`
Change user password by using code from email.

Status Codes:
- 204 No Content – Password has been reset.
- default – Operation failure response.

`GET /user/{userId}/`
Get user by id.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Found user.
- default – Operation failure response.

`POST /user/{userId}/`
Update user with new parameters.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Updated workspace.
- default – Operation failure response.

`DELETE /user/{userId}/`
Strip personal information from user, quit all groups and block any access to this user.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – User has been soft-deleted.
- default – Operation failure response.

`GET /user/{userId}/public/`
Get user public profile by id.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Found user.
- default – Operation failure response.

`POST /user/{userId}/login/password/`
Change user password by using temporary code or old password.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – Password has been changed.
- default – Operation failure response.

`POST /user/{userId}/mfa/email-code/`
Configure email-code multi-factor authentication.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – Multi-factor authentication has been configured.
- default – Operation failure response.

`DELETE /user/{userId}/login/tokens/`
Revoke all issues tokens for specified user.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – All tokens have been revokes. It is required to re-authenticate after this call.
- default – Operation failure response.

`POST /user/{userId}/login/api-key/`
Add API key login to user.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – List of workspaces.
- default – Operation failure response.

DELETE /user/{userId}/login/api-key/
   Delete API key login from user.

        Parameters:
- **userId** (*string*) – Id of user.

        Query Parameters:
- **id** (*string*) – Id of UserLogin with API key. (Required)

        Status Codes:
- 204 No Content – API key has been deleted and no longer valid.
- default – Operation failure response.

POST /user/{userId}/invitations/{invitationId}/
   Accept invitation.

        Parameters:
- **userId** (*string*) – Id of user.
- **invitationId** (*string*) – Id of invitation.

        Status Codes:
- 204 No Content – Invitation has been accepted.
- default – Operation failure response.

DELETE /user/{userId}/invitations/{invitationId}/
   Decline invitation.

        Parameters:
- **userId** (*string*) – Id of user.
- **invitationId** (*string*) – Id of invitation.

        Status Codes:
- 204 No Content – Invitation has been dismissed.
- default – Operation failure response.

Workspace

GET /workspace/
   Get all available workspaces.

        Query Parameters:
- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
- **query** (*string*) – Any value to search in workspace name.

        Status Codes:
- 200 OK – List of workspaces.
- default – Operation failure response.

GET /workspace/my/
   Get current user's workspaces.

        Status Codes:
- 200 OK – List of known workspaces.
- default – Operation failure response.

GET /workspace/{workspaceId}/
   Get workspace by id.

        Parameters:
- **workspaceId** (*string*) – Id of workspace.

        Status Codes:
- 200 OK – Found Workspace.
- default – Operation failure response.

POST /workspace/{workspaceId}/
   Update workspace with new parameters.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Updated workspace.
- default – Operation failure response.

PUT /workspace/{workspaceId}/administrators/
  Promote member to workspace administrators.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Query Parameters:
- **memberUserId** (*string*) – Member user id. (Required)

Status Codes:
- 204 No Content – Member has been promoted to administrator.
- default – Operation failure response.

DELETE /workspace/{workspaceId}/administrators/
  Demote member from workspace administrators.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Query Parameters:
- **memberUserId** (*string*) – Member user id. (Required)

Status Codes:
- 204 No Content – Member has been demoted from administrator.
- default – Operation failure response.

GET /workspace/{workspaceId}/members/
  Get workspace members.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Workspace members.
- default – Operation failure response.

GET /workspace/
  Get all available workspaces.

Query Parameters:
- **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
- **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
- **query** (*string*) – Any value to search in workspace name.

Status Codes:
- 200 OK – List of workspaces.
- default – Operation failure response.

GET /workspace/my/
  Get current user's workspaces.

Status Codes:
- 200 OK – List of known workspaces.
- default – Operation failure response.

GET /workspace/{workspaceId}/
  Get workspace by id.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Found Workspace.
- default – Operation failure response.

POST /workspace/{workspaceId}/

Update workspace with new parameters.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Updated workspace.
- default – Operation failure response.

`PUT /workspace/{workspaceId}/administrators/`
Promote member to workspace administrators.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Query Parameters:
- **memberUserId** (*string*) – Member user id. (Required)

Status Codes:
- 204 No Content – Member has been promoted to administrator.
- default – Operation failure response.

`DELETE /workspace/{workspaceId}/administrators/`
Demote member from workspace administrators.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Query Parameters:
- **memberUserId** (*string*) – Member user id. (Required)

Status Codes:
- 204 No Content – Member has been demoted from administrator.
- default – Operation failure response.

`GET /workspace/{workspaceId}/members/`
Get workspace members.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Workspace members.
- default – Operation failure response.

WorkspaceQuota

`POST /workspace/{workspaceId}/quota-usage/`
Get workspace quota usage.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Found Workspace.
- default – Operation failure response.

`POST /workspace/{workspaceId}/quota-usage/`
Get workspace quota usage.

Parameters:
- **workspaceId** (*string*) – Id of workspace.

Status Codes:
- 200 OK – Found Workspace.
- default – Operation failure response.

Project

`GET /project/`
Get all available projects.

Query Parameters:	• **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
	• **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
	• **query** (*string*) – Any value to search in project name.
Status Codes:	• 200 OK – List of projects.
	• default – Operation failure response.

PUT /project/
Create new project.

| Status Codes: | • 200 OK – Created project. |
| | • default – Operation failure response. |

GET /project/my/
Get current user's projects.

| Status Codes: | • 200 OK – List of projects. |
| | • default – Operation failure response. |

GET /project/{projectId}/
Get project by id.

Parameters:	• **projectId** (*string*) – Id of project. Project are located within workspace.
Status Codes:	• 200 OK – Found Project.
	• default – Operation failure response.

POST /project/{projectId}/
Update project with new parameters.

Parameters:	• **projectId** (*string*) – Id of project. Project are located within workspace.
Status Codes:	• 200 OK – Updated project.
	• default – Operation failure response.

DELETE /project/{projectId}/
Delete project and all related data.

Parameters:	• **projectId** (*string*) – Id of project. Project are located within workspace.
Status Codes:	• 204 No Content – Project has been deleted.
	• default – Operation failure response.

PUT /project/{projectId}/branch/
Create branch in project.

Parameters:	• **projectId** (*string*) – Id of project. Project are located within workspace.
Status Codes:	• 204 No Content – Branch has been created.
	• default – Operation failure response.

PUT /project/{projectId}/branch/{branchName}/
Push branch content into another branch in this project.

| Parameters: | • **branchName** (*string*) – Name of branch. Branches are located inside Project. |
| | • **projectId** (*string*) – Id of project. Project are located within workspace. |

Status Codes:
- 204 No Content – Branch has been updated.
- default – Operation failure response.

POST /project/{projectId}/branch/{branchName}/
Update branch in project.

Parameters:
- **branchName** (*string*) – Name of branch. Branches are located inside Project.
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Branch has been updated.
- default – Operation failure response.

DELETE /project/{projectId}/branch/{branchName}/
Delete branch in project.

Parameters:
- **branchName** (*string*) – Name of branch. Branches are located inside Project.
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Branch has been deleted.
- default – Operation failure response.

POST /project/{projectId}/workspace/
Transfer project form one workspace to another.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Project has been transfered.
- default – Operation failure response.

PUT /project/{projectId}/members/
Invite another user into project.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Invite has been sent.
- default – Operation failure response.

DELETE /project/{projectId}/members/
Expel another user from project.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Query Parameters:
- **memberUserId** (*string*) – Member user id. (Required)

Status Codes:
- 204 No Content – Invite has been sent.
- default – Operation failure response.

POST /project/{projectId}/permissions/
Update project permissions.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Project permissions has been updated.
- default – Operation failure response.

GET /project/
Get all available projects.

Query Parameters:	• **skip** (*integer*) – Number of elements to skip during paging. Aka offset or start.
	• **take** (*integer*) – Number of elements to take during paging. Aka limit or count.
	• **query** (*string*) – Any value to search in project name.
Status Codes:	• 200 OK – List of projects.
	• default – Operation failure response.

PUT /project/
  Create new project.

> **Status Codes:**
> - 200 OK – Created project.
> - default – Operation failure response.

GET /project/my/
  Get current user's projects.

> **Status Codes:**
> - 200 OK – List of projects.
> - default – Operation failure response.

GET /project/{projectId}/
  Get project by id.

> **Parameters:**
> - **projectId** (*string*) – Id of project. Project are located within workspace.
>
> **Status Codes:**
> - 200 OK – Found Project.
> - default – Operation failure response.

POST /project/{projectId}/
  Update project with new parameters.

> **Parameters:**
> - **projectId** (*string*) – Id of project. Project are located within workspace.
>
> **Status Codes:**
> - 200 OK – Updated project.
> - default – Operation failure response.

DELETE /project/{projectId}/
  Delete project and all related data.

> **Parameters:**
> - **projectId** (*string*) – Id of project. Project are located within workspace.
>
> **Status Codes:**
> - 204 No Content – Project has been deleted.
> - default – Operation failure response.

PUT /project/{projectId}/branch/
  Create branch in project.

> **Parameters:**
> - **projectId** (*string*) – Id of project. Project are located within workspace.
>
> **Status Codes:**
> - 204 No Content – Branch has been created.
> - default – Operation failure response.

PUT /project/{projectId}/branch/{branchName}/
  Push branch content into another branch in this project.

> **Parameters:**
> - **branchName** (*string*) – Name of branch. Branches are located inside Project.
> - **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Branch has been updated.
- default – Operation failure response.

POST /project/{projectId}/branch/{branchName}/
Update branch in project.

Parameters:
- **branchName** (*string*) – Name of branch. Branches are located inside Project.
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Branch has been updated.
- default – Operation failure response.

DELETE /project/{projectId}/branch/{branchName}/
Delete branch in project.

Parameters:
- **branchName** (*string*) – Name of branch. Branches are located inside Project.
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Branch has been deleted.
- default – Operation failure response.

POST /project/{projectId}/workspace/
Transfer project form one workspace to another.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Project has been transfered.
- default – Operation failure response.

PUT /project/{projectId}/members/
Invite another user into project.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Invite has been sent.
- default – Operation failure response.

DELETE /project/{projectId}/members/
Expel another user from project.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Query Parameters:
- **memberUserId** (*string*) – Member user id. (Required)

Status Codes:
- 204 No Content – Invite has been sent.
- default – Operation failure response.

POST /project/{projectId}/permissions/
Update project permissions.

Parameters:
- **projectId** (*string*) – Id of project. Project are located within workspace.

Status Codes:
- 204 No Content – Project permissions has been updated.
- default – Operation failure response.

Membership

GET /membership/packages/

Get all membership packages.

> **Status Codes:**
> - 200 OK – List of all membership packages.
> - default – Operation failure response.

`GET /membership/packages/`
Get all membership packages.

> **Status Codes:**
> - 200 OK – List of all membership packages.
> - default – Operation failure response.

Billing

`GET /billing/{userId}/account/`
Get billing account by id.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Found billing account.
> - default – Operation failure response.

`POST /billing/{userId}/account/`
Update billing information

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Updated billing account.
> - default – Operation failure response.

`POST /billing/{userId}/contact-request/`
Request contact from sales representative.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 204 No Content – An contact request has been made.
> - default – Operation failure response.

`GET /billing/{userId}/payment/status/`
Get status of payment for subscription for workspace.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Query Parameters:**
> - **sessionOrInvoiceId** (*string*) – Payment session Id or invoice Id. (Required)
>
> **Status Codes:**
> - 200 OK – Status of payment session or invoice.
> - default – Operation failure response.

`POST /billing/{userId}/payment/status/`
Start subscription session for workspace.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Payment action response.
> - default – Operation failure response.

`POST /billing/{userId}/payment/`
Make payment for selected invoice.

> **Parameters:**
> - **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Payment action response.
- default – Operation failure response.

POST /billing/{userId}/payment/upcoming/

Get prorated upcoming payment information.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Upcoming payment information
- default – Operation failure response.

POST /billing/{userId}/customer-portal/

Get url of customer portal for user if available.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Portal url or none.
- default – Operation failure response.

POST /billing/notification/

Accept notification from payment gate.

Status Codes:
- 204 No Content – Notification has been accepted.
- default – Operation failure response.

GET /billing/{userId}/account/

Get billing account by id.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Found billing account.
- default – Operation failure response.

POST /billing/{userId}/account/

Update billing information

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 200 OK – Updated billing account.
- default – Operation failure response.

POST /billing/{userId}/contact-request/

Request contact from sales representative.

Parameters:
- **userId** (*string*) – Id of user.

Status Codes:
- 204 No Content – An contact request has been made.
- default – Operation failure response.

GET /billing/{userId}/payment/status/

Get status of payment for subscription for workspace.

Parameters:
- **userId** (*string*) – Id of user.

Query Parameters:
- **sessionOrInvoiceId** (*string*) – Payment session Id or invoice Id. (Required)

Status Codes:
- 200 OK – Status of payment session or invoice.
- default – Operation failure response.

POST /billing/{userId}/payment/status/
Start subscription session for workspace.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Payment action response.
> - default – Operation failure response.

POST /billing/{userId}/payment/
Make payment for selected invoice.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Payment action response.
> - default – Operation failure response.

POST /billing/{userId}/payment/upcoming/
Get prorated upcoming payment information.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Upcoming payment information
> - default – Operation failure response.

POST /billing/{userId}/customer-portal/
Get url of customer portal for user if available.

> **Parameters:**
> - **userId** (*string*) – Id of user.
>
> **Status Codes:**
> - 200 OK – Portal url or none.
> - default – Operation failure response.

Search

POST /search/
Search for users, projects, workspaces by specified keyword.

> **Status Codes:**
> - 200 OK – Search result with found search items.
> - default – Operation failure response.

POST /search/
Search for users, projects, workspaces by specified keyword.

> **Status Codes:**
> - 200 OK – Search result with found search items.
> - default – Operation failure response.

ResourceStorage

PUT /resourceStorage/
Create resource.

> **Query Parameters:**
> - **name** (*string*) – Filename or name of resource. (Required)
> - **purpose** (*string*) – Use purpose of resource. (Required)
> - **mediaType** (*string*) – Media type of resource.
>
> **Status Codes:**
> - 200 OK – Created resource id.
> - default – Operation failure response.

Overview

`GET /resourceStorage/{resourceId}/`
Get resource metadata by id.

    Parameters:
- **resourceId** (*string*) – Id of resource.

    Status Codes:
- 200 OK – Found resource.
- default – Operation failure response.

`DELETE /resourceStorage/{resourceId}/`
Delete resource by id.

    Parameters:
- **resourceId** (*string*) – Id of resource.

    Status Codes:
- 204 No Content – Resource has been deleted.
- default – Operation failure response.

`GET /resourceStorage/{resourceId}/data/`
Get resource binary data by id.

    Parameters:
- **resourceId** (*string*) – Id of resource.

    Status Codes:
- 200 OK – Found resource.

`PUT /resourceStorage/`
Create resource.

    Query Parameters:
- **name** (*string*) – Filename or name of resource. (Required)
- **purpose** (*string*) – Use purpose of resource. (Required)
- **mediaType** (*string*) – Media type of resource.

    Status Codes:
- 200 OK – Created resource id.
- default – Operation failure response.

`GET /resourceStorage/{resourceId}/`
Get resource metadata by id.

    Parameters:
- **resourceId** (*string*) – Id of resource.

    Status Codes:
- 200 OK – Found resource.
- default – Operation failure response.

`DELETE /resourceStorage/{resourceId}/`
Delete resource by id.

    Parameters:
- **resourceId** (*string*) – Id of resource.

    Status Codes:
- 204 No Content – Resource has been deleted.
- default – Operation failure response.

`GET /resourceStorage/{resourceId}/data/`
Get resource binary data by id.

    Parameters:
- **resourceId** (*string*) – Id of resource.

    Status Codes:
- 200 OK – Found resource.

Context

`GET /context/`
Get page context.

<table>
<tr><td align="right">Query Parameters:</td><td>

- **projectName** (*string*) – Project name of current page.
- **branchName** (*string*) – Branch name of current page.

</td></tr>
<tr><td align="right">Status Codes:</td><td>

- 200 OK – Page's context related properties.
- default – Operation failure response.

</td></tr>
</table>

`GET /context/`
Get page context.

<table>
<tr><td align="right">Query Parameters:</td><td>

- **projectName** (*string*) – Project name of current page.
- **branchName** (*string*) – Branch name of current page.

</td></tr>
<tr><td align="right">Status Codes:</td><td>

- 200 OK – Page's context related properties.
- default – Operation failure response.

</td></tr>
</table>

Notifications

`GET /notification/`
Subscribe on notifications from server. This is WebSocket endpoint, any non 'Upgrade' requests will fail.

> **Status Codes:**
> - 101 Switching Protocols – WebSocket upgrade were sucessful.

`GET /notification/`
Subscribe on notifications from server. This is WebSocket endpoint, any non 'Upgrade' requests will fail.

> **Status Codes:**
> - 101 Switching Protocols – WebSocket upgrade were sucessful.

Troubleshooting

`PUT /app/log/`
Log specified message on server. Used internally while standalone-hosted.

> **Status Codes:**
> - 204 No Content – URL has been opened (or ignored).
> - default – Operation failure response.

Basic Navigation and User Interface Overview

The UI consists of a left-side menu displaying all schemas of the game data, a middle working area with a dashboard/document list or document form, and a headline on the top with the project name and settings button. Depending on the installation, the UI may also include a user menu.

Dashboard

The dashboard is a central hub in the game data's user interface that provides quick access to frequently used features. It includes quick action buttons, such as creating a new schema, export, import, as well as a list of recently visited documents. Additionally, in the case of a web application, the dashboard may display the presence of online members who are currently working in the same project.

Document Collection

The document collection page is place where user can view a list of all the documents of a specified schema. This page allows users to filter, sort, and customize the list to their liking, making it easier to find the specific document they need.

Document Form

The document form page provides a specific edit form for a selected document. Here, users can view, edit, and save their game data documents in a structured and organized manner. The form allows users to input data into fields that correspond to the schema's properties. The document form page provides a user-friendly interface for updating and modifying game data.

See also

- Creating Document Type (Schema)
- Filling Documents
- Publishing Game Data
- Generating Source Code

Creating Document Type (Schema)

Schema

Schema is essential for organizing and defining game data in a structured framework. In the context of game data modeling, a schema serves as a blueprint or template that establishes the structure and properties of a particular type of data entity in a game. It defines the columns or fields that represent the various attributes of the entity, similar to how a table has columns or a spreadsheet has cells.

Enemy		
Name → Text	**HP** → Number	**Attack** → Number
Zombie	10	5
Skeleton	5	2

← Schema

Benefits of Structured Data

Data Organization

The organization of game data into logical entities and attributes is facilitated by the schema. This blueprint or template defines the structure, properties, and relationships of different entities and attributes within the game data, ensuring efficient storage, retrieval, and management.

Data Validation

The integrity and adherence to predefined rules of the game data are ensured through the validation capabilities of the schema. Constraints and validations, such as data types, allowed values, and dependencies, can be defined, preventing errors and inconsistencies in the game data.

Data Consistency

Consistency across the game data is achieved through the standardized structure and rules provided by the schema. It enforces a consistent naming convention, attribute definitions, and relationships between entities, thereby enhancing coherence and simplifying collaboration.

Data Interoperability

Interoperability and integration with external systems or tools are facilitated by a well-defined schema. By establishing a common language and structure, the schema enables seamless data exchange and collaboration with localization tools, analytics platforms, and asset pipelines.

Analyzing Game Requirements
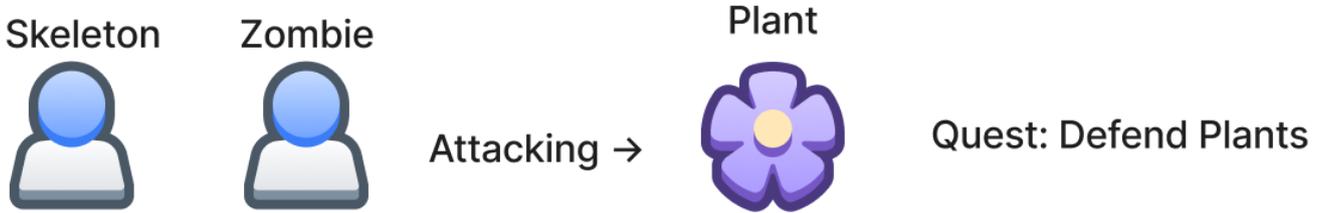
This step involves analyzing the game requirements to understand the design and functionality of the game. It includes studying the game design document and identifying key features, gameplay mechanics, and data elements that need to be captured and represented in the game.

Identifying Schemas and Relationships
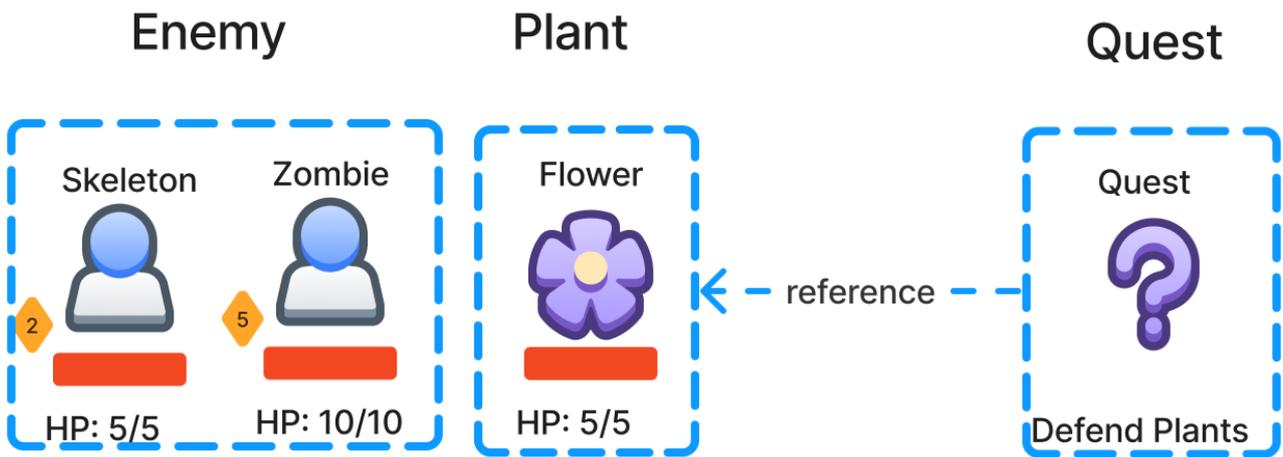
In this step, schemas and their relationships within the game are identified. Schemas can be objects, characters, locations, items, quests, or any other significant element in the game. Relationships define how these entities are connected or interact with each other.

Defining Schemas and Properties
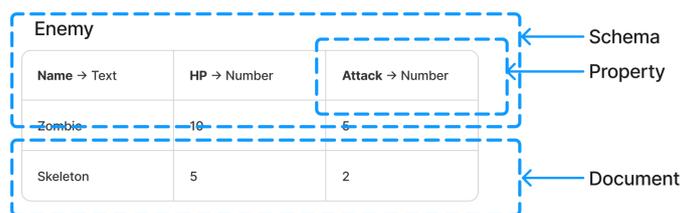
This step involves defining schemas to represent the structure and properties of the game data. A schema serves as a blueprint or template for a specific type of data entity, specifying its properties, attributes, and relationships. Properties describe the characteristics and attributes of an entity, such as its name, description, stats, or any other relevant information.

All Data Types

Date

The `Date` data type is used to store dates in ISO 8601 format, which includes the year, month, day, and time with UTC time zone. This data type is particularly useful for storing information about events that occur on specific dates or for tracking the age of entities. Since dates are stored with UTC time zone, the data can be consistently interpreted across different time zones.

C# Type

```
System.DateTime
```

Uniqueness

May NOT be checked for uniqueness.

Format

```
yyyy-MM-ddTHH:mm:ss.fffZ
```

Example

```
"2017-12-27T00:00:00.000Z"

// it is better not to store dates before this mark for compatibility reasons
"1970-01-01T00:00:00.000Z"
```

Document

The `Document` data type in game data schema is used to represent complex structures. A document can contain multiple properties of different data types, including other documents or document collections, allowing for hierarchical data modeling. It is important to note that the lifetime of sub-documents is tied to the lifetime of the parent document, meaning that any changes (e.g. deletion) to the parent document will affect all of its sub-documents.

C# Type

```
class
```

Uniqueness

May NOT be checked for uniqueness.

Example

For example, in a **Dialog**, each node can be a `Document` with dialog text, response options, and actions that occur after a response is chosen. Each response option can be a sub-document that is another **Dialog** node.

```
{
    "Text": "Welcome to the game. What's your name?",
    "Options": [
        {
            "Text": "My name is John.",
            "Options": [
                {
                    "Text": "Hello John! What brings you here?",
                    "Options": [
                        {
                            "Text": "I'm looking for adventure.",
                            "Action": "dialog.GiveQuestAndEnd()"
                        },
                        {
```

```json
                              "Text": "I'm on a mission.",
                              "Options": [/* ... */]
                        }
                  ]
            }
      ]
   },
   {
      "Text": "I prefer not to say.",
      "Action": "dialog.End()"
   }
   ]
}
```

Document Collection

The DocumentCollection data type is used to store an array of sub-documents, which are used to represent complex structures. It is important to note that the lifetime of sub-documents is tied to the lifetime of the parent document, meaning that any changes (e.g. deletion) to the parent document will affect all of its sub-documents.

C# Type

   `ReadOnlyList{T}` where `T` is Schema

Uniqueness

   May NOT be checked for uniqueness.

Size

   May be limited in number of items. 0 - no limit.

Example

One example use case for `DocumentCollection` is storing a list of items in a game, such as a chest and its contents. Each item in the chest could be represented by a sub-document containing information such as reference to an item and its quantity.

```json
{
   "Name": "Silver Chest",
   "Loot": [
      {
         "Item": { "Id": "Sword" },
         "Quantity": 1
      },
      {
         "Item": { "Id": "Silver" },
         "Quantity": 100
      }
   ]
}
```

Formula

Formula data type is a way to store and use C# expressions inside game data. It allows game developers to perform calculations based on certain inputs and parameters that are not known until runtime. A formula can be any valid C# expression that returns a value of any supported data type.

Formulas are typically used in situations where there are complex calculations involved, such as determining the damage a weapon does based on various factors like the target's resistance and the type of attack being used. By storing these calculations as formulas, game developers can easily modify and tweak them without having to recompile the entire game code.

During runtime, formulas can be evaluated using the values of any other properties or data types that are passed to formula as arguments. This allows for a great deal of flexibility in designing game mechanics and balancing gameplay.

Overview

For example, a formula for calculating the damage a weapon does to a target could be stored as follows:

```
(weaponPower * (1.0 - targetResistance)) * attackMultiplier
```

This formula takes the weapon power, subtracts the target's resistance, and then multiplies the result by an attack multiplier. The resulting value is the final amount of damage the weapon does to the target.

C# Type

 class

Uniqueness

 May NOT be checked for uniqueness.

Example

```
"target.HP < 100"
"x != 0"
"target.DoDamage(100)"
"(weapon.Damage / target.DamageResistance) / 2"
```

Integer

The `Integer` data type is a whole number data type that is limited to 64 bits. It is used to represent integers without a fractional component. It can be used in cases where you need to store a large range of positive or negative whole numbers, such as in-game currency or player levels.

Unlike the Number data type, integers do not have any precision caveats since they do not store decimal values. Therefore, they are suitable for calculations that require exact values.

C# Type

 System.SByte, System.Int16, System.Int32, System.Int64

Uniqueness

 May be checked for uniqueness.

Size

 32 or 64bit

Example

```
0
-1
100
```

Localized Text

The `LocalizedText` data type is used to store text that needs to be displayed in multiple languages. Unlike the Text data type, the LocalizedText data type allows the storage of the same text in multiple languages. It supports the whole range of UTF symbols, just like the Text data type. The `LocalizedText` data type is essential for games that require localization support, and it makes it easy for game developers to manage text that needs to be displayed in multiple languages.

C# Type

 LocalizedString or System.String

Uniqueness

 May NOT be checked for uniqueness.

Size

 May be limited in number of characters. 0 - no limit.

Example

```
{"en-US": "Hello", "fr-FR": "Bonjour"}
```

Overview

Logical

The `Logical` data type is used to represent boolean values, i.e., values that can be either true or false. It is commonly used in game development to represent various settings, options, or conditions.

For example, a game designer may use a `Logical` data type to represent whether a particular game feature is enabled or disabled. The `Logical` data type can be used in combination with control flow statements, such as conditional statements or loops, to determine the behavior of the game.

The MultiPickList data type can also be used to represent boolean values, but it allows the selection of multiple options instead of just two. This can be useful for representing more complex options or flags that require multiple selections. However, if the options are limited to just two, it is recommended to use the Logical data type for clarity and simplicity.

C# Type
> System.Boolean

Uniqueness
> May be checked for uniqueness.

Example

```
true
false
```

Multi-Pick List

The `MultiPickList` data type is used when you want to allow the selection of multiple values from a predefined list of options. It is similar to the PickList data type, but it allows for multiple selections.

`MultiPickList` is particularly useful when you want to replace several Logical properties that have a related meaning with a single property. For example, instead of having three separate properties to indicate if a item can be broken, disassembled, or sold, you can use a MultiPickList with the options "CanBeBreaken," "CanBeDisassembled," and "CanBeSold."

C# Type
> `enum` based on System.SByte, System.Int16, System.Int32, System.Int64

Uniqueness
> May be checked for uniqueness.

Size
> 32 or 64bit

Example

```
1 // internaly stored as integers
"Apple" // string values also valid
```

Number

The `Number` data type is used to represent decimal numbers. It conforms to the IEEE 754 floating-point standard and can represent both positive and negative numbers, as well as zero. However, due to the limitations of the floating-point representation, precision may be lost when performing certain arithmetic operations. Therefore, it is recommended to use the Integer data type for financial calculations and other scenarios that require high precision.

Some use cases for the `Number` data type include representing quantities, such as the count of an items or the amount of gold reward in the chest, or representing percentages, such as the chance of an event occurring. It can also be used to represent measurements, such as the height of a character or the length of a weapon.

When working with `Numbers` in game data, it is important to ensure that the precision is appropriate for the use case. Additionally, it may be necessary to round numbers to a certain number of decimal places to avoid displaying unnecessarily precise values to players.

C# Type
> System.Single or System.Double

63

Uniqueness

    May be checked for uniqueness.

Size

    32 or 64bit

Example

```
3.14
0.21
-3.14
```

Pick List

`PickList` is a data type used to define a list of pre-defined options for a property. It allows the user to select only one option from the given list. The options can be defined as a string, and the list can contain any number of options.

`PickList` data type is commonly used to define properties such as gender, language, or country, where there are a limited number of options to choose from. It provides a convenient way to standardize the data, and also helps to prevent errors or inconsistencies in the data.

For example, in a game where the player can choose a character class, the `PickList` data type can be used to define the available options, such as "Warrior," "Mage," or "Rogue." This ensures that the player can only choose from the available options and helps to prevent invalid inputs.

C# Type

    `enum` based on `System.SByte, System.Int16, System.Int32, System.Int64`

Uniqueness

    May be checked for uniqueness.

Size

    32 or 64bit

Example

```
1 // internaly stored as integers
"Apple" // string values also valid
```

Reference

The `Reference` data type allows the creation of non-embedding relationships between documents. A reference is essentially a pointer to another document, using that document's `Id` as a key. This allows for easier linking between related documents, without having to embed one document inside another.

When using a `Reference`, the referenced documents are not stored within the parent document, but rather as references to their respective locations. This can be useful when dealing with large, complex data sets where it's more efficient to reference data than to embed it. Additionally, this data type can help enforce data integrity by ensuring that references to other documents are valid.

For example, in a game, a **Chest** with loot table might have a reference to a specific inventory **Item** document, rather than having the entire **Item** embedded inside the **Chest** document. This makes it easier to manage the loot separately of items and maintain the relationship between the **Chest**, loot table and and the **Item**.

C# Type

    `Reference{T}` or `T` where `T` is Schema

Uniqueness

    May NOT be checked for uniqueness.

Example

```
{ "Id": "Sword" }
"Sword" // just raw Id is also accepted
```

Reference Collection

The `ReferenceCollection` data type is used to create non-embedded relationships between documents. It allows for referencing multiple documents of the same type from within another document.

When using a `ReferenceCollection`, the referenced documents are not stored within the parent document, but rather as references to their respective locations. This can be useful when dealing with large, complex data sets where it's more efficient to reference data than to embed it. Additionally, this data type can help enforce data integrity by ensuring that references to other documents are valid.

For example, a game might have a collection of quests, each of which references a collection of objectives. The objectives might be stored in a separate collection for purpose rof e-use, and can be referenced by the quest document using the `ReferenceCollection` data type.

C# Type

> `ReadOnlyList{T}` or `ReadOnlyList{Reference{T}}` where `T` is Schema

Uniqueness

> May NOT be checked for uniqueness.

Size

> May be limited in number of items. 0 - no limit.

Example

```
[{ "Id": "Sword" }, { "Id": "Gold" }]
["Sword", "Gold"] // just raw Ids are also accepted
```

Text

The `Text` data type is used to store simple text values in game data. Unlike the LocalizedText data type, `Text` does not have support for multiple translations of the same text. Instead, it allows for the storage of any UTF symbol in a single language. This data type is useful for fields that do not require localization, such as character names, item descriptions, or game lore.

C# Type

> `System.String`

Uniqueness

> May be checked for uniqueness (case sensitive for uniqueness purposes).

Size

> May be limited in number of characters. 0 - no limit.

Example

```
"Hello world!"
```

Time

The `Time` data type in game data is equivalent to the `TimeSpan` data type in C#. It is used to store a duration or a time interval, such as the time it takes to complete a task or the length of a cutscene in a game. The `Time` data type is represented as a string in the format `HH:mm:ss`, where HH is the number of hours, mm is the number of minutes, and ss is the number of seconds.

For example, if a task takes 2 hours and 30 minutes to complete, the `Time` data type value would be `02:30:00`.

C# Type

> `System.TimeSpan`

Uniqueness

> May NOT be checked for uniqueness.

Format

> `[DD.]HH:mm:ss` or `<number-of-seconds>`

Overview

```
"02:30:00" // 2 hours and 30 minutes
"1.00:00:00" // 1 day
60 // 60 seconds
120 // two minutes

"-00:30:00" // could be negative
```

Selecting the proper data type is important in order to ensure that data is correctly and efficiently stored and used in the game. Each data type has its own specific purpose and characteristics, which should be considered when choosing the appropriate type for a given property.

For example, if a property needs to store a text value, the **Text** data type would be appropriate. If the text needs to be translated into multiple languages, the **LocalizedText** data type would be the best choice.

Similarly, if a property needs to store a numeric value, the **Number** or **Integer** data types would be appropriate depending on the type of number being stored.

The **PickList** and **MultiPickList** data types are useful for properties that have a limited set of values, such as a list of game items or character classes.

The **Document** and **DocumentCollection** data types are useful for storing complex data that may contain multiple fields or properties.

Ultimately, selecting the proper data type ensures that game data is properly structured, and helps to prevent errors and inconsistencies in the game.

Table with example

Data Type	Description	Example
Text	A line of text.	"Hello, world!"
LocalizedText	A localized text.	{"en-US": "Hello", "fr-FR": "Bonjour"}
Logical	A true/false value.	True
Time	A time span.	"1.00:00:00"
Date	A specific date.	"2017-12-27T00:00:00.000Z"
Number	A decimal number.	3.14
Integer	A whole number.	42
PickList	A list of pre-defined values.	"Red"
MultiPickList	A list of pre-defined values that can have multiple selections.	"Apple, Banana, Cherry"
Document	An embedded document.	{ "Id": "Sword", "Name": "Rusty Sword" }
DocumentCollection	A collection of embedded documents.	[{ "Id": "Sword", "Name": "Rusty Sword" }]
Reference	A reference to another document.	{ "Id": "Sword"}
ReferenceCollection	A collection of references to other documents.	[{ "Id": "Sword" }]
Formula	A C#-like expression used to calculate something.	"target.HP < 100"

See also

- Implementing Inheritance

- Filling Documents
- Publishing Game Data
- Generating Source Code

Filling Documents

Once the game data structure has been defined, there are several methods available for creating and populating game entities. One option is to import game data from other sources, such as tables or JSON files. Another option is to generate data using external tools and import it into the editor. Finally, data can be added gradually as development progresses using the game data editor.

Importing JSON files

JSON files can be imported via the user interface by following these steps:

1. Navigate to the document collection page.
2. Click on `Actions → Import...`.
3. Select the JSON file and follow the steps in the import wizard.

See structure requirements.

Exporting to Spreadsheet and Importing Back

To export game data to a spreadsheet for editing and then import it back, follow these steps:

1. Navigate to the document collection page.
2. Click on `Actions → Export To → Spreadsheet (.xlsx)` to export the data to a spreadsheet file.
3. Open the downloaded file and make the necessary edits.
4. **Import the modified data back into the system:**

   a. Drag and drop the edited file onto the document collection page.
   b. Alternatively, click on `Actions → Import...` and follow the steps in the import wizard to select and import the modified file.

Adding New Document

To create a new document using the user interface, follow these steps:

1. Navigate to the document collection page.
2. Click on the `Create` button.
3. Fill in the required fields in the form provided.
4. Click `Save` to save the new document.

See also

- Publishing Game Data
- Generating Source Code

Generating Source Code

The process of generating source code allows game data to be used inside a game. This process involves specifying the language (e.g. C#) and various generation parameters/optimizations. It can be done from both the project's dashboard user interface and the command-line interface (CLI).

Features

Overview

Feature	C#	TypeScript	C++ (UE)	Haxe
JSON Format	x	x	x	x
MessagePack Format	x	x	x	x
Language Switch	x	x	x	x
Patching	x	x	x	x
Formulas	x	x		
By Unique Value Indexing	x			x

Using Project's Dashboard UI

To generate source code from the dashboard, follow these steps:

1. Go to the dashboard of the project where you want to generate source code.
2. Click on the *Generate Source Code* button.
3. Choose the language you want to generate the source code in.
4. Specify any generation parameters required.
5. Click on the *Generate* button to initiate the process.
6. Download archive file with generated source code.

Using Command-Line Interface (CLI)

To generate source code from the CLI, follow these steps:

1. Open the command-line interface.
2. Navigate to the game data's directory.
3. Use the `GENERATE <SOURCECODE>` command to generate the source code, specifying the target language and any generation parameters require.

Example

```
dotnet charon GENERATE CSHARPCODE --dataBase "c:\my app\gamedata.json" --namespace "MyGame.P
```

- Use the `--outputDirectory` parameter to specify the location where generated files will be saved.
- Use the `--namespace` and `--gameDataClassName` parameters to adjust the signature of generated classes.
- Use the `--splitFiles` parameter to generate multiple files instead of one large one.
- Use the `--clearOutputDirectory` parameter to clear the output directory from generated files when re-generating source code.

Once the process is complete, the generated source code will be available at `--outputDirectory`.

See also

- Publishing Game Data
- Working with Source Code (C# 4.0)
- Working with Source Code (C# 7.3)
- Working with Source Code (TypeScript)
- Working with Source Code (UE C++)
- **Working with Source Code (Haxe)**

- Command Line Interface (CLI)
- GENERATE CSHARPCODE Command
- GENERATE TYPESCRIPTCODE Command
- GENERATE UECPP Command
- GENERATE HAXE Command

Implementing Inheritance

Inheritance is a familiar tool for programmers when working with shared behavior or data. Unfortunately, it is not natively supported in Charon. However, you can achieve similar functionality using alternative approaches.

For example, imagine you have three different document types: `Armor`, `Weapon`, and `Shield`. You want to include all these items in a `Shop` list of sellable goods. In traditional inheritance, you could create a base type to unify these documents and refer to it.

Armor
```
{
  Id: ITEM_ARMOR
  Name: Iron Armor,
  Price: 10,
  AP: 1
}
```

Weapon
```
{
  Id: ITEM_SWORD,
  Name: Iron Sword,
  Price: 10,
  Damage: 2
}
```

Shield
```
{
  Id: ITEM_SHIELD,
  Name: Wooden Shield,
  AP: 10,
  BashDamage: 1
}
```

Shop
```
{
  Id: SHOP_LIST_1
  Items: [
    ???
  ]
}
```

Without inheritance, here are three possible approaches:

1. Composition

Extract the shared data from all sellable item types into a separate document type called `Item`. Each sellable document (e.g., `Armor`, `Weapon`, `Shield`) should include an embedded `Item` document containing store-relevant information. In the `Shop` list of sellable goods, you can store references to `Item` documents.

Armor
```
{
  Id: ARMOR
  Item: {                    Item
    Id: ITEM_ARMOR
    Name: Iron Armor,
    Price: 10,
  }
  AP: 1
}
```

Weapon
```
{
  Id: SWORD,
  Item: {
    Id: ITEM_SWORD,
    Name: Iron Sword,
    Price: 10,
  }
  Damage: 2
}
```

Shield
```
{
  Id: SHIELD,
  Item: {
    Id: ITEM_SHIELD,
    Name: Wooden Shield,
    Price: 12
  }
  AP: 10
  BashDamage: 1
}
```

Shop
```
{
  Id: SHOP_LIST_1
  Items: [
    { Id: ITEM_ARMOR },
    { Id: ITEM_SWORD },       References
    { Id: ITEM_SHIELD },
  ]
}
```
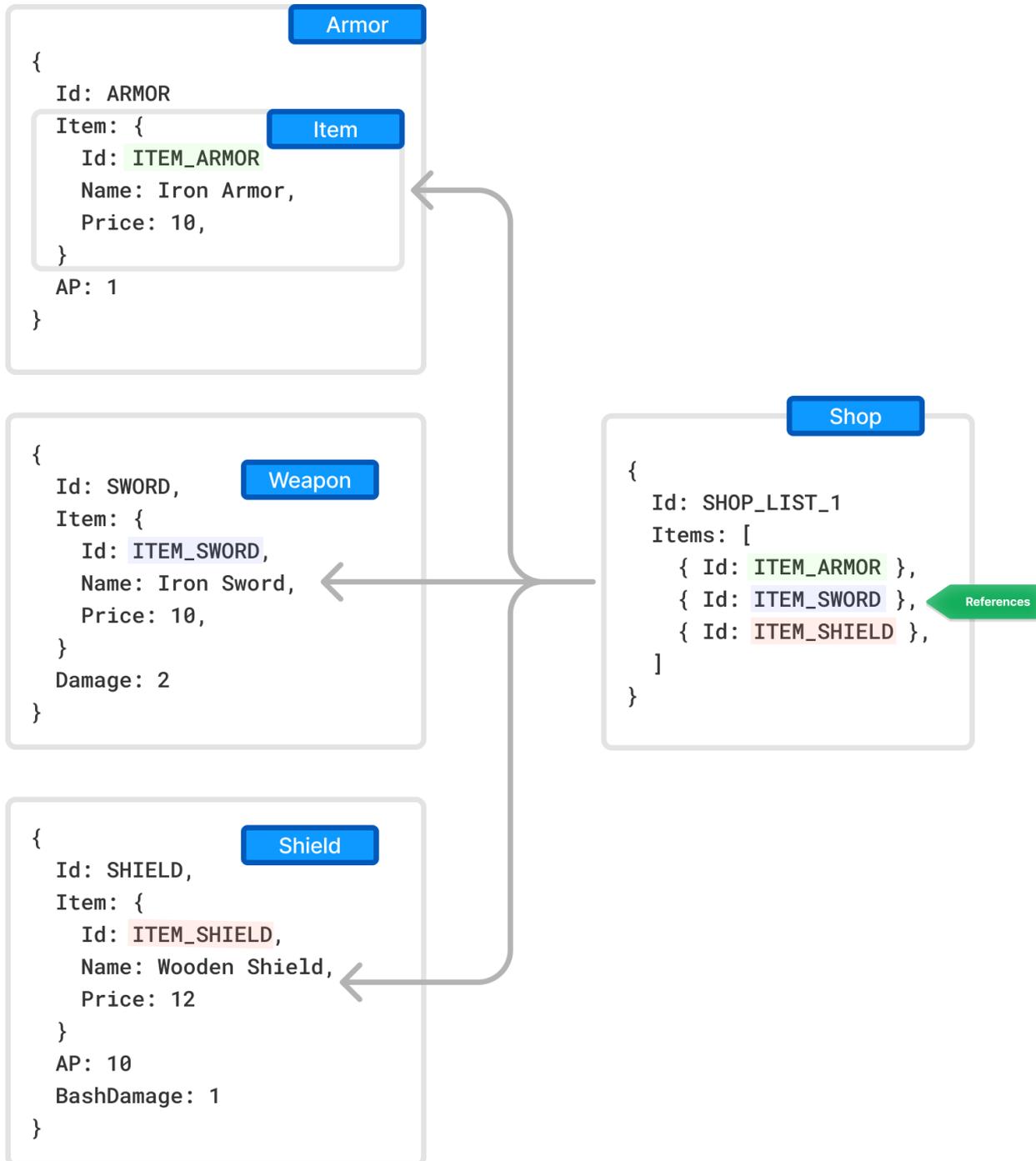
2. Merging

Alternatively, combine the three document types (`Armor`, `Weapon`, and `Shield`) into a single document type called `Item`. This document will contain fields for all three original types, along with an additional `Type` field to specify the item's category. The `Shop` list can then store references to these unified `Item` documents.

3. Aggregation

As a less elegant alternative, introduce a `ShopItem` type with fields referencing all possible document types (`Armor`, `Weapon`, and `Shield`). In each `ShopItem` document, only one of these fields will be filled, depending on the item's type. The `Shop` list can then reference `ShopItem` documents.

Armor

```
{
  Id: ITEM_ARMOR
  Name: Iron Armor,
  Price: 10,
  AP: 1
}
```

Weapon

```
{
  Id: ITEM_SWORD,
  Name: Iron Sword,
  Price: 10,
  Damage: 2
}
```

Shield

```
{
  Id: ITEM_SHIELD,
  Name: Wooden Shield,
  AP: 10,
  BashDamage: 1
}
```

Shop

```
{
  Id: SHOP_LIST_1
  Items: [
    {                            ShopItem
      Id: 1
      Armor: null,
      Weapon: ITEM_SWORD,    ◀ Reference
      Shield: null,
    },
    {
      Id: 2
      Armor: ITEM_ARMOR,
      Weapon: null,
      Shield: null,
    }
  ]
}
```
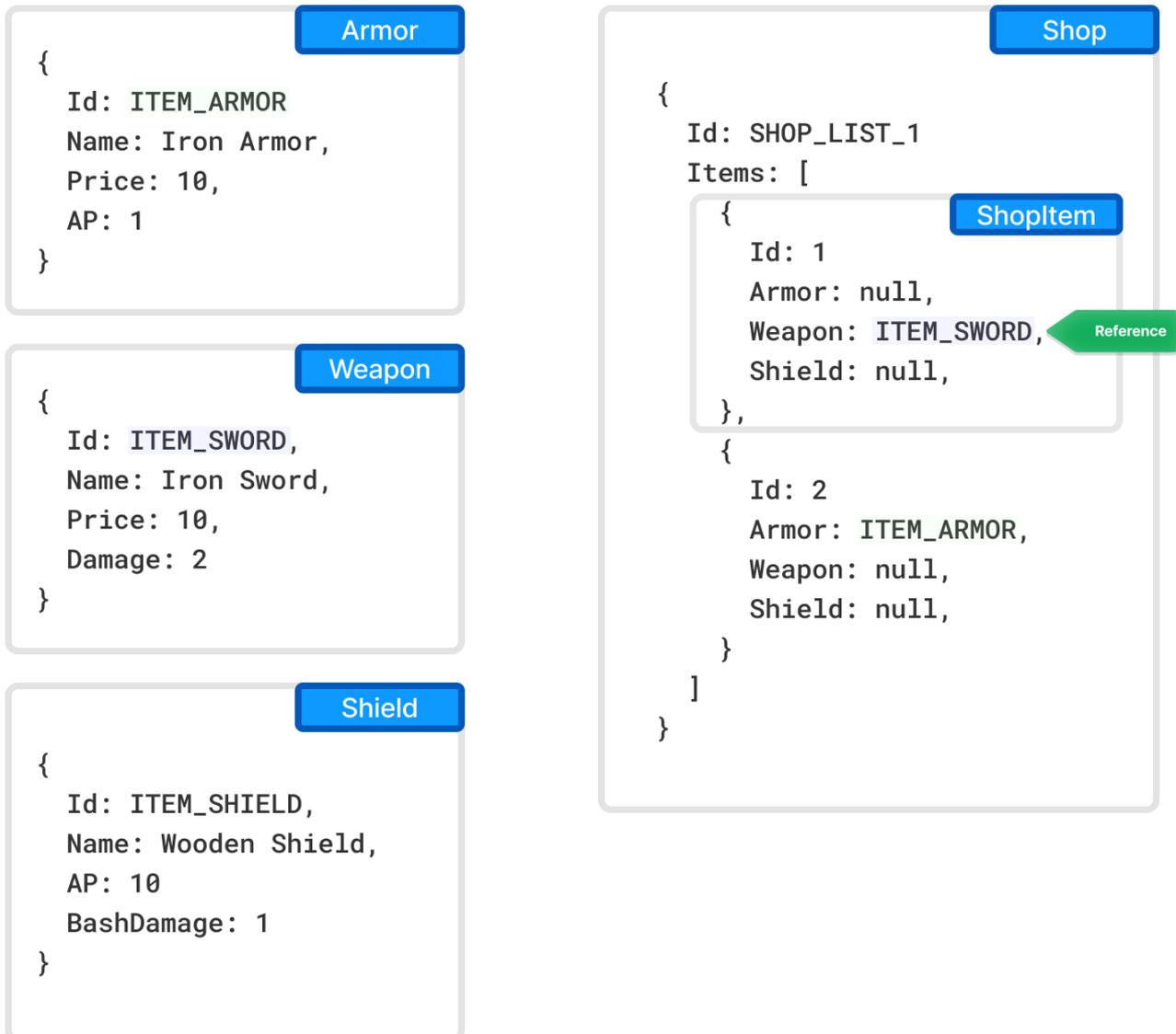
Conclusion

Each of these methods has its trade-offs in terms of simplicity, flexibility, and performance. Choosing the right approach depends on your application's requirements and the expected complexity of your data model.

See also

- Creating Document Type (Schema)
- Filling Documents
- Publishing Game Data
- Generating Source Code

Publishing Game Data

The publication process is a crucial step in preparing game data for usage inside the game. This process involves removing unused data, unused localization, and exporting data in a supported format - `JSON` or `MessagePack`. This documentation will provide an overview of how to perform the publication process from both the project's dashboard user interface and the command-line interface (CLI).

Using Project's Dashboard UI

To perform the publication process from the project's dashboard UI, please follow the steps below:

1. Navigate to the project's dashboard page.
2. Click on the *Publish* link.
3. Choose the format you want to export your data in - JSON or Message Pack.
4. Select the language(s) you want to publish.
5. Click on the *Finish* button to initiate the publication process.
6. Download the file.

Using Command-Line Interface (CLI)

To perform the publication process from the CLI, please follow the steps below:

1. Open the command-line interface.
2. Navigate to the game data's directory.
3. Use the `DATA EXPORT` command to publish the game data.

Example

```
dotnet charon DATA EXPORT --dataBase ".\gamedata.json" --mode publication --languages {en-US
```

- Use the `--languages` parameter to specify the language(s) you want to publish. Or omit parameter to publish all languages.
- Use the `--outputFormat` parameter to specify the export format - `json` or `msgpack`.

See also

- Generating Source Code
- Working with Source Code (C# 4.0)
- Working with Source Code (C# 7.3)
- Working with Source Code (TypeScript)
- Command Line Interface (CLI)
- DATA EXPORT Command

Working with Source Code (C# 4.0)

> ### Warning
>
> This is deprecated code generator and shouldn't be used in new projects.

Accessing game data during runtime is possible by utilizing the generated source code.

Overview

This section provides examples using default class names, but it is possible to customize class names during the source code generation process. Additionally, this customization allows to avoid naming collisions with existing code.

Loading Game Data

The following C# code creates `GameData` class and loads your game data into memory.

```csharp
using System.IO;

var fileStream = File.OpenRead("gamedata.json");
var gameData = new GameData(fileStream, GameData.Format.Json);
fileStream.Dispose();
```

The file `gamedata.json` could be published game data or original database file (.gdjs or .gdmp).

Accessing Documents

You can access your documents as a list:

```csharp
var characters = gameData.GetCharacters() // -> ReadOnlyList<Character>
var characters = gameData.GetCharacters(onlyRoot: true) // -> ReadOnlyList<Character>
```

Or you can access specific documents by their `Id` or *Unique* properties:

```csharp
var character = gameData.GetCharacter(characterId); // -> Character
var character = gameData.GetCharacterByName(characterName); // -> Character
```

`Settings` schemas are accessed by name:

```csharp
var resetTime = gameData.LootSettings.ResetTime; // -> TimeSpan
```

Formulas

Formulas are executed with Invoke method:

```csharp
var reward = gameData.LootSettings.RewardFormula.Invoke()  // -> int
```

Formula's parameters are passed as arguments of `Invoke` method.

Generated Code Extensions

When generating source code for game data, the resulting C# classes are declared as partial. This means that the classes can be extended by the programmer to add custom functionality.

For example, let's say that you have generated a `GameData` class for your game data. This class contains properties and methods for accessing and manipulating the data. However, you want to add some custom functionality to this class, such as a method for getting specific documents by criteria.

To do this, you can create a new C# file and declare a partial class with the same name as the generated `GameData` class. You can then define your custom method in this class, and it will be merged with the generated class at compile time.

Here is an example of how this could look:

In this example, the `GameData` class is declared as partial, and two partial classes are defined with the same name: one generated by the source code generation process and one containing custom code added by the programmer.

By using partial classes in this way, you can extend the functionality of the generated classes without modifying the generated code directly. This allows you to keep your custom code separate from the generated code, making it easier to maintain and update your game data classes over time.

There is also two extension points on `GameData` class:

```csharp
partial void OnBeforeInitialize(); // Called after loading the data into lists and dictionar
partial void OnInitialize(); // Called after loading and prepping all data.
```

73

Working with Source Code (C# 7.3)

Accessing game data during runtime is possible by utilizing the generated source code.

This section provides examples using default class names, but it is possible to customize class names during the source code generation process. Additionally, this customization allows to avoid naming collisions with existing code.

Loading Game Data

The following C# code creates `GameData` class and loads your game data into memory.

```csharp
using System.IO;

var fileStream = File.OpenRead("RpgGameData.gdjs"); // or .json
var gameData = new GameData(fileStream, new Formatters.GameDataLoadOptions {
  Format = Formatters.Format.Json,
  // Patches = new [] { patchStream1, patchStream2, ... }
});
fileStream.Dispose();
```

The file `RpgGameData.gdjs` could be published game data or original database file (.gdjs or .gdmp).

Accessing Documents

You can access your documents as a list:

```csharp
var allHeroes = gameData.AllHeroes.AsList // -> IReadOnlyList<Hero>
var heroes = gameData.Heroes.AsList // -> IReadOnlyList<Hero>
```

Or you can access specific documents by their `Id` or *Unique* properties:

```csharp
var heroById = gameData.AllHeroes.Get(heroId); // -> Hero
var heroByName = gameData.AllHeroes.ByName().Get(heroName); // -> Hero
```

`Settings` schemas are accessed by name:

```csharp
var startingHeroes = gameData.StartingSet.Heroes; // -> IReadOnlyList<Hero>
```

Formulas

Formulas are executed with Invoke method:

```csharp
var reward = gameData.LootSettings.RewardFormula.Invoke()  // -> int
```

Formula's parameters are passed as arguments of `Invoke` method.

Generated Code Extensions

When generating source code for game data, the resulting C# classes are declared as partial. This means that the classes can be extended by the programmer to add custom functionality.

For example, let's say that you have generated a `GameData` class for your game data. This class contains properties and methods for accessing and manipulating the data. However, you want to add some custom functionality to this class, such as a method for getting specific documents by criteria.

To do this, you can create a new C# file and declare a partial class with the same name as the generated `GameData` class. You can then define your custom method in this class, and it will be merged with the generated class at compile time.

Here is an example of how this could look:

Overview

In this example, the `GameData` class is declared as partial, and two partial classes are defined with the same name: one generated by the source code generation process and one containing custom code added by the programmer.

By using partial classes in this way, you can extend the functionality of the generated classes without modifying the generated code directly. This allows you to keep your custom code separate from the generated code, making it easier to maintain and update your game data classes over time.

There is also two extension points on `GameData` class:

```
partial void OnInitialize(); // Called after loading and prepping all data.
```

See also

- Generating Source Code
- GENERATE CSHARPCODE Command

Working with Source Code (Haxe)

Accessing game data during runtime is possible by utilizing the generated source code.

This section provides examples using default class names, but it is possible to customize class names during the source code generation process. Additionally, this customization allows to avoid naming collisions with existing code.

Loading Game Data

The following Haxe code creates `GameData` class and loads your game data into memory.

```
import GameData;
import Formatters;
import haxe.io.Path;
import sys.io.File;

var input = File.read("RpgGameData.gdjs"); // or .json
var options = new GameDataLoadOptions();
options.format = GameDataFormat.Json;
options.leaveInputsOpen = false;
// options.patches <-- put patches here
var gameData = new GameData(input, options);
```

The file `RpgGameData.gdjs` could be published game data or original database file (.gdjs or .gdmp).

Accessing Documents

You can access your documents as a list:

```
var allHeroes = gameData.heroesAll.list // -> ReadOnlyArray<Hero>
var heroes = gameData.heroes.list // -> ReadOnlyArray<Hero>
```

Or you can access specific documents by their `id` or *Unique* properties:

`Settings` schemas are accessed by name:

```
var startingHeroes = gameData.startingSet.heroes; // -> ReadOnlyArray<Hero>
```

Formulas

Formulas are currently not supported.

See also

- Generating Source Code
- GENERATE HAXE Command

Working with Source Code (Type Script)

Accessing game data during runtime is possible by utilizing the generated source code.

This section provides examples using default class names, but it is possible to customize class names during the source code generation process. Additionally, this customization allows to avoid naming collisions with existing code.

Loading Game Data

The following Type Script code creates `GameData` class and loads your game data into memory.

```typescript
import { GameData } from './game.data';
import { Formatters } from './formatters';

// Node.js
import { readFileSync } from 'fs';
const gameDataStream = readFileSync(gameDataFilePath);

// Blob or File
const gameDataStream = gameDataFileBlob.arrayBuffer();

// XMLHttpRequest (XHR)
// gameDataRequest.responseType -> "arraybuffer"
const gameDataStream = gameDataRequest.response;

const gameData = new GameData(gameDataStream, {
  format: Formatters.GameDataFormat.Json,
  // patches: [patchStream1, patchStream2, ...]
});
```

The content of `gameDataStream` could be published game data or original database file (.gdjs or .gdmp).

Accessing Documents

You can access your documents as a list:

```typescript
let heroes = gameData.heroesAll; // all heroes from all documents -> readonly Hero[]
let heroes = gameData.heroes; // heroes only from root collection -> readonly Hero[]
```

Or you can access specific documents by their `Id` or *Unique* properties:

```typescript
let hero = gameData.heroesAll.find(heroId); // -> Hero | undefined
let hero = gameData.heroesAll.withOtherKey('Name').find(heroName); // -> Hero | undefined
```

`Settings` schemas are accessed by name:

```typescript
let resetTime = gameData.lootSettings.resetTime; // -> TimeSpan
```

Formulas

Formulas inherit the `Function` type and can be invoked as-is or with `invoke` method:

```typescript
var reward = gameData.lootSettings.rewardFormula()  // -> number
// or
var reward = gameData.lootSettings.rewardFormula.invoke()  // -> number
```

Formula's parameters are passed as arguments of `invoke` method.

Any non-game data related types are imported from `formula.known.types.ts, which should be created by the developer and have all required types exported. Here is an example of a `formula.known.types.ts` file:

```typescript
import { MyFormulaContext } from '../my.formula.context';

// example of MyFormulaContext type.
export MyFormulaContext;
```

Overview

```
// example of Assets.Scripts.CheckContext.
export namespace Assets.Scripts {
  export class CheckContext {
    myField: string;
  }
}
```

See also

- Generating Source Code
- GENERATE TYPESCRIPTCODE Command

Working with Source Code (UE C++)

> **Warning**
>
> The source code for Unreal Engine requires a plugin to be installed to function. If you get compilation errors, make sure the plugin is installed and enabled.

Accessing game data during runtime is possible by utilizing the generated source code.

This section provides examples using default class names, but it is possible to customize class names during the source code generation process. Additionally, this customization allows to avoid naming collisions with existing code.

Loading Game Data

The following C++ code creates `UGameData` class and loads your game data into memory.

```cpp
IFileManager& FileManager = IFileManager::Get();

const FString GameDataFilePath = TEXT("./RpgGameData.gdjs");  // or .json
const TUniquePtr<FArchive> GameDataStream = TUniquePtr<FArchive>(FileManager.CreateFileReade

UGameData* GameData = NewObject<UGameData>();

FGameDataLoadOptions Options;
Options.Format = EGameDataFormat::Json;
// Options.Patches.Add(PatchStream1);
// Options.Patches.Add(PatchStream2);
// ...

if (!GameData->TryLoad(GameDataStream.Get(), Options))
{
  // Handle failure
}
```

The file `RpgGameData.gdjs` could be published game data or original database file (.gdjs or .gdmp).

Accessing Documents

You can access your documents as a list:

```cpp
auto AllHeroes = GameData->AllHeroes // -> TMap<string, UHero>
auto Heroes = GameData->Heroes // -> TMap<string, UHero>
```

`Settings` schemas are accessed by name:

Overview

```
auto StartingHeroes = GameData->StartingSet.Heroes; // -> TMap<string, UHero>
```

Formulas

Formulas are currently not supported.

See also

- Generating Source Code
- GENERATE UECPP Command

Command Line Interface (CLI)

Most of Charon functionality could be accessed via CLI commands. The application itself uses the getops syntax. You should be familiar with terminal on your OS to fully tap potential of CLI.

Installation

Download and install NET 8+.

Option 1: dotnet tool (recommended)

The easiest way to install is to use the infrastructure provided by the dotnet tool.

```
# install charon globally
dotnet tool install -g dotnet-charon

# install charon in current working directory
dotnet tool install dotnet-charon --local --create-manifest-if-needed
```

To update current tool use following commands:

```
# update global tool
dotnet tool update --global dotnet-charon

# update local tool
dotnet tool update dotnet-charon --local
```

Option 2: Bootstrap scripts

Alternatively, you can use one of two bootstrap scripts:

- RunCharon.bat (Windows)
- RunCharon.sh (Linux, MacOS)

Both scripts require the dotnet tool to be included in the system PATH. The scripts handle the installation of the Charon tool and ensure it stays up to date.

Windows

```
mkdir Charon
cd Charon
curl -O https://raw.githubusercontent.com/gamedevware/charon/main/scripts/bootstrap/RunCharo

.\RunCharon.bat DATA EXPORT --help
#               ^
#       your command goes here
```

Linux, MacOS

```
mkdir Charon
cd Charon
```

```
curl -O https://raw.githubusercontent.com/gamedevware/charon/main/scripts/bootstrap/RunCharon

chmod +x ./RunCharon.sh

./RunCharon.sh DATA EXPORT --help
#                  ^
#       your command goes here
```

Command Syntax

Commands have the following syntax:

```
dotnet charon COMMAND --parameterName <parameter-value>

# parameters can have more than one value.
# Use space to separate values
dotnet charon EXPORT --schemas Item Armor "Project Settings" Quest

# if your value contains a space, put it inside the quotation marks.
# Escape characters and other rules depend on the OS you are running.
dotnet charon "c:\my application\my path.txt"

# some parameters don't require a value (e.g. flag).
dotnet charon VERSION --verbose
```

Absolute and relative paths

When running commands, it's crucial to be aware of whether you are using absolute or relative paths to files.

1. **Absolute Path**: An absolute path defines a file or directory's location in relation to the root directory. In Linux and macOS, it starts from the root `/`, while in Windows, it begins with a drive letter (like `C:\`).

   - Example for Linux/macOS: `/usr/local/bin`

   - Example for Windows: `C:\Program Files\mono`

2. **Relative Path**: A relative path references a file or directory in relation to the current working directory, without starting with a root slash or drive letter.

   - Example: If currently in `/home/user/Documents`, a file in `/home/user/Documents/Projects` would have the relative path `Projects/FileName`.
   - **Windows Command Prompt**: Paths use backslashes (`\`). Absolute paths start with a drive letter (like `C:\Users\Name`), while relative paths use the file name or paths like `subfolder\file.txt`.
   - **macOS/Linux Terminal**: Paths are denoted with forward slashes (`/`). Absolute paths begin from the root (`/`), and relative paths use `./` for the current directory or `../` to go up one level.

Getting Help Text

To display list of available commands add *–help* or */?*.

```
dotnet charon --help

#> Usage: dotnet charon <action> [--<param> || (--<param> <paramValue> ...) ...]
#>
#> Verbs:
#>  DATA        Data manipulation actions.
#>  GENERATE    Code generation actions.
#>  VERSION     Print version.

dotnet charon DATA EXPORT --help

#> Usage:
```

```
#>   DATA EXPORT --dataBase <URI> [--schemas [<TEXT>]] [--properties [<TEXT>]] [--languages
#>            ] [--outputFormat <TEXT>] [--outputFormattingOptions [<TEXT>]] [--mode <EXP
#>            TEXT>]]
```

Apply Patch

Applies patch created with DATA CREATEPATCH command to a game data.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA APPLYPATCH --dataBase "c:\my app\gamedata.json" --input "c:\my app\gameda

# remote game data
dotnet charon DATA APPLYPATCH --dataBase "https://charon.live/view/data/My_Game/develop/" --
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of *–dataBase* being URL.
`--input`	Path to a file with patch to apply. Alternatively, you can use Standart Input or *URL <remote_input_output>*.

```
# standart input (default)
--input in
--input con

# absolute path (windows)
--input "c:\my app\gamedata_patch.json"

# absolute path (unix)
--input "/user/data/gamedata_patch.json"

# relative path (universal)
--input "./gamedata_patch.json"

# remote location (HTTP)
--input "http://example.com/gamedata_patch.json"

# remote location with authentication (FTP)
--input "ftp://user:password@example.com/gamedata_patch.json"
```

Overview

`--inputFormat`	Format of imported data.

```
# Auto-detect by extension (default)
--inputFormat auto

# JSON
--inputFormat json

# BSON
--inputFormat bson

# Message Pack
--inputFormat msgpack

# XML (removed in 2025.1.1)
--inputFormat xml
```

`--inputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Create Backup

Backs up game data to a specified file. Saved data could be later used with DATA RESTORE command.
Also this command can be used to convert game data into different format.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA BACKUP --dataBase "c:\my app\gamedata.json" --output "c:\my app\backup.ms

# remote game data
dotnet charon DATA BACKUP --dataBase "https://charon.live/view/data/My_Game/develop/" --outp
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of –*dataBase* being URL.

Overview

| `--output` | Path to a backup file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL. |

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\backup.json"

# absolute path (unix)
--output "/user/data/backup.json"

# relative path (universal)
--output "./backup.json"

# remote location (HTTP)
--output "http://example.com/backup.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/backup.json"
```

| `--outputFormat` | Format of backed up data. |

```
# JSON (default)
--outputFormat json

# Message Pack
--outputFormat msgpack
```

| `--outputFormattingOptions` | Additional options for specified format. |

This command supports universal parameters.

Output

The back up data follows the general game data structure.

Create Document

Creates a new document. For a bulk creations use DATA IMPORT command with `--mode create`.
Only the first document from the `--input` will be processed.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA CREATE --dataBase "c:\my app\gamedata.json" --schema Item --input "c:\my

# remote game data
dotnet charon DATA CREATE --dataBase "https://charon.live/view/data/My_Game/develop/" --sche
```

Parameters	
`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of *–dataBase* being URL.
`--schema`	Name or identifier of the type (schema) of the new document.

```
# name
--schema Item

# id
--schema 55a4f32faca22e191098f3d9
```

`--input`	Path to a file with document. Alternatively, you can use Standart Input or URL.

```
# standart input (default)
--input in
--input con

# absolute path (windows)
--input "c:\my app\item.json"

# absolute path (unix)
--input "/user/data/item.json"

# relative path (universal)
--input "./item.json"

# remote location (HTTP)
--input "http://example.com/item.json"

# remote location with authentication (FTP)
--input "ftp://user:password@example.com/item.json"
```

`--inputFormat`	Format of imported data.

```
# Auto-detect by extension (default)
--inputFormat auto

# JSON
--inputFormat json

# BSON
--inputFormat bson

# Message Pack
--inputFormat msgpack

# XML (removed in 2025.1.1)
--inputFormat xml
```

`--inputFormattingOptions`	Additional options for specified format.

Overview

| `--output` | Path to a created document file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL. |

```
# standart output
--output out
--output con

# standart error
--output err

# null device (default)
--output null

# absolute path (windows)
--output "c:\my app\created_item.json"

# absolute path (unix)
--output /user/data/created_item.json

# relative path (universal)
--output "./created_item.json"

# remote location (HTTP)
--output "http://example.com/created_item.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/created_item.json"
```

| `--outputFormat` | Format of created data. |

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

| `--outputFormattingOptions` | Additional options for specified format. |

This command supports universal parameters.

Input Data Schema

The data you input should follow this schema (recommended):

```
{
  "Collections": {
    "<Schema-Name>": [
      {
        // <Document>
      }
    ]
  }
}
```

This schema is also accepted:

Overview

```
{
  "<Schema-Name>": [
    {
      // <Document>
    }
  ]
}
```

A list of documents is accepted:

```
[
  {
    // <Document>
  }
]
```

And single document too:

```
{
  // <Document>
}
```

Output

Outputs the created document with all the edits that were made to make it conform to the schema.

```
{
  "Id": "Sword"

  /* rest of properties of created document */
}
```

Create Patch

Outputs the differences between two game datas as a file that can be used later to DATA APPLYPATCH to another game data.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA CREATEPATCH --dataBase "c:\my app\gamedata.json" --input "c:\my app\gamed

# remote game data
dotnet charon DATA CREATEPATCH --dataBase "https://charon.live/view/data/My_Game/develop/" -
```

Parameters

--dataBase1	Absolute or relative path to a first game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase1 "c:\my app\gamedata.json"

# remote server
--dataBase1 "https://charon.live/view/data/My_Game/develop/"
```

`--dataBase2`	Absolute or relative path to a second game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase2 "c:\my app\gamedata.json"

# remote server
--dataBase2 "https://charon.live/view/data/My_Game/develop/"
```

`--output`	Path to a patch file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\gamedata_patch.json"

# absolute path (unix)
--output /user/data/gamedata_patch.json

# relative path (universal)
--output "./gamedata_patch.json"

# remote location (HTTP)
--output "http://example.com/gamedata_patch.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/gamedata_patch.json
```

`--outputFormat`	Format of exported data.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

`--outputFormattingOptions`	Additional options for specified format.
`--credentials`	This parameter sets the API key used to access *BOTH* remote servers. If this is not suitable, consider downloading the data locally and running this command on local files instead.

This command supports universal parameters.

Delete Document

Deletes a document. For a bulk deletion use DATA IMPORT command with `--mode delete`.

- CLI Installation

Overview

- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA DELETE --dataBase "c:\my app\gamedata.json" --schema Item --id "Sword"

# remote game data
dotnet charon DATA DELETE --dataBase "https://charon.live/view/data/My_Game/develop/" --sche
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials	The API key used to access remote server in case of –dataBase being URL.
--schema	Name or identifier of the type (schema) of deleting document.

```
# name
--schema Item

# id
--schema 55a4f32faca22e191098f3d9
```

--id	Identifier of deleting document.

```
# text
--id Sword

# number
--id 101
```

Overview

`--output`	The path to a file where the *deleted document* should be placed. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output
--output out
--output con

# standart error
--output err

# null device (default)
--output null

# absolute path (windows)
--output "c:\my app\deleted_item.json"

# absolute path (unix)
--output /user/data/deleted_item.json

# relative path (universal)
--output "./deleted_item.json"

# remote location (HTTP)
--output "http://example.com/deleted_item.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/deleted_item.json"
```

`--outputFormat`	Format for deleted document.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

`--outputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Output

Outputs the deleted document in its state at the time of deletion.

```
{
  "Id": "Sword"

  /* rest of properties of deleted document */
}
```

Export Data

Exports documents into a file.

- CLI Installation
- Commands Reference
- Universal Parameters

Overview

- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA EXPORT --dataBase "c:\my app\gamedata.json" --schemas Character --output

# remote game data
dotnet charon DATA EXPORT --dataBase "https://charon.live/view/data/My_Game/develop/" --sche
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials	The API key used to access remote server in case of *–dataBase* being URL.
--schemas	A list of types of documents (schemas) to export. By default all schemas *EXCEPT* metadata are exported.

- Use space to separate multiple schemas.

- You can use wildcards (*) at the beginning and end of names.

- You can use identifiers in {} instead of names.

- You can exclude certain names by using an exclamation mark (!) at the beginning of their names.

```
# schema name
--schemas Character
--schemas Character Item

# all (default)
--schemas *

# masks
--schemas Char*
--schemas *Modifier
--schemas *Mod*

# schema id
--schemas {18d4bf318f3c49688087dbed}

# negation
--schemas Char* !Character
--schemas !*Item*

# excluding system schemas (Schema, SchemaProperty, ProjectSettings
--schemas ![system]
```

`--properties`	A list of properties or property types to export. By default all properties are exported.

- *Id* property always included

- Use space to separate multiple properties.

- You can use wildcards (*) at the beginning and end of names.

- You can use identifiers in {} instead of names.

- You can exclude certain names by using an exclamation mark (!) at the beginning of their names.

- You can use data type in [] instead of names.

`--languages`	List of languages to keep in exported data. Language's english name is used or language tag (BCP 47).

Use DATA I18N LANGUAGES to get list of used languages.

- Use space to separate multiple languages

- You can use wildcards (*) at the beginning and end of names.

- You can use LCID or CultureInfo.Name in {} instead of the name.

- You can exclude certain names by using an exclamation mark (!) at the beginning of their names.

```
# language tag (BCP 47)
--languages {en-US}

# language name
--languages "Spanish (Spain)"

# language name mask
--languages Spanish*

# language LCID
--languages {3082}

# negation and masks
--languages !Spanish*
--languages Spanish* !{es-Es}
```

`--mode`	Export mode controls stripping and inclusion rules for exported data.

```
# (default)
--mode normal

--mode publication
--mode extraction
--mode localization
```

normal

Export all specified documents defined in *–schemas.* This mode ensures that the exported graph of documents remains valid by including any necessary additional documents to avoid any broken references.

publication

Same as *–mode normal*, but all non-essential data will be stripped. The result of the export can be safely loaded within the game with the generated code.

extraction

Export only the specified *–schemas* without exporting any referenced documents. In this mode, the exported graph of documents may contain broken references. It is recommended to use the import *–mode safeupdate* when importing this data back.

localization

Same as *–mode extraction* but only `LocalizedText` properties are exported.

`--output`	Path to a exported data file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\document.json"

# absolute path (unix)
--output /user/data/document.json

# relative path (universal)
--output "./document.json"

# remote location (HTTP)
--output "http://example.com/document.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/document.json"
```

`--outputFormat`	Format of exported data.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml

# XLSX Spreadsheet
--outputFormat xlsx
```

`--outputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Output

The exported data follows the general game data structure, but omits *ToolsVersion*, *RevisionHash*, and *ChangeNumber* when the export mode is **not** set to publication.

```json
{
  "Collections":
  {
    "Character":
    [
      {
        "Id": "Knight"

        /* rest of properties of document */
      },
      {
        "Id": "Templar"

        /* rest of properties of document */
      },
      // ...
    ]
  }
}
```

Modifying Exported Data with *yq*

The exported data can be accessed or modified using the *yq* tool, a lightweight and portable command-line YAML, JSON, and XML processor. *yq* uses *jq*-like syntax and supports common operations for manipulating structured data.

To use *yq* with exported JSON data:

1. **Install `yq`**: Follow the installation instructions from the official *yq* documentation: https://mikefarah.gitbook.io/yq/.

2. **Query Data**: Use *yq* to query specific fields or values from the exported JSON file.

   ```
   # Query a specific field
   yq '.Collections.Character[0].name' characters.json
   ```

3. **Modify Data**: Use *yq* to update or add fields in the exported JSON file.

4. **Convert Formats**: *yq* can also convert between JSON, YAML, and other supported formats.

```
# Convert JSON to YAML
yq -o=yaml characters.json > characters.yaml
```

For more advanced usage, refer to the *yq* documentation: https://mikefarah.gitbook.io/yq/.

Find Document

Seaches for a document.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA FIND --dataBase "c:\my app\gamedata.json" --schema Character --id John

# remote game data
dotnet charon DATA FIND --dataBase "https://charon.live/view/data/My_Game/develop/" --schema
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of *–dataBase* being URL.
`--schema`	Name or identifier of the type (schema) of document.

```
# name
--schema Item

# id
--schema 55a4f32faca22e191098f3d9
```

`--id`	Identifier of document.

```
# text
--id Sword

# number
--id 101
```

Overview

| --output | Path to a found document file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL. |

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\document.json"

# absolute path (unix)
--output /user/data/document.json

# relative path (universal)
--output "./document.json"

# remote location (HTTP)
--output "http://example.com/document.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/document.json"
```

| --outputFormat | Format of exported data. |

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

| --outputFormattingOptions | Additional options for specified format. |

This command supports universal parameters.

Output

Outputs the found document.

```
{
  "Id": "John"

  /* rest of properties of found document */
}
```

Add Translation Languages

Add translation languages to specified game data.

- CLI Installation
- Universal Parameters

Overview

- Commands Reference

Command

```
# local game data (windows)
dotnet charon DATA I18N ADDLANGUAGE --dataBase "c:\my app\gamedata.json" --languages "es-ES"

# remote game data
dotnet charon DATA I18N ADDLANGUAGE --dataBase "https://charon.live/view/data/My_Game/develo
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.
	
	`# local file`
	`--dataBase "c:\my app\gamedata.json"`
	
	`# remote server`
	`--dataBase "https://charon.live/view/data/My_Game/develop/"`

--languages	The list of languages to add. Values are language tags (BCP 47) separated by space..

This command supports universal parameters.

Export Translated Data

Export text that can be translated into a file.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA I18N EXPORT --dataBase "c:\my app\gamedata.json" --schemas Character --so

# remote game data
dotnet charon DATA I18N EXPORT --dataBase "https://charon.live/view/data/My_Game/develop/" -
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.
	
	`# local file`
	`--dataBase "c:\my app\gamedata.json"`
	
	`# remote server`
	`--dataBase "https://charon.live/view/data/My_Game/develop/"`

--credentials	The API key used to access remote server in case of *–dataBase* being URL.

--schemas	A list of types of documents (schemas) to export. By default all schemas *EXCEPT* metadata are exported.

- Use space to separate multiple schemas.

- You can use wildcards (*) at the beginning and end of names.

- You can use identifiers in {} instead of names.

- You can exclude certain names by using an exclamation mark (!) at the beginning of their names.

```
# schema name
--schemas Character
--schemas Character Item


# all (default)
--schemas *


# masks
--schemas Char*
--schemas *Modifier
--schemas *Mod*


# schema id
--schemas {18d4bf318f3c49688087dbed}


# negation
--schemas Char* !Character
--schemas !*Item*


# excluding system schemas (Schema, SchemaProperty, ProjectSettings)
--schemas ![system]
```

--sourceLanguage	Source (original) language for translation. Value is language tag (BCP 47).
	Use DATA I18N LANGUAGES to get list of used languages.

```
# it is used as <source> in XLIFF
--sourceLanguage en-US
```

--targetLanguage	Target language for translation. Value is language tag (BCP 47).

```
# it is used as <target> in XLIFF
--targetLanguage es-ES
```

Overview

`--output`	Path to a file to which data will be exported. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\input.json"

# absolute path (unix)
--output /user/data/input.json

# relative path (universal)
--output "./input.json"

# remote location (HTTP)
--output "http://example.com/input.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/input.json"
```

`--outputFormat`	Format of exported data.

```
# XLIFF v2 (default)
--outputFormat xliff
--outputFormat xliff2

# XLIFF v1
--outputFormat xliff1

# XSLX Spreadsheet
--outputFormat xslx

# JSON
--outputFormat json
```

`--outputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Output

The exported data follows the general game data structure, but omits *ToolsVersion*, *RevisionHash*, and *ChangeNumber* fields.

Importing Translated Data

Import translated text from a specified file into game data.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Overview

```
# local game data (windows)
dotnet charon DATA I18N IMPORT --dataBase "c:\my app\gamedata.json" --input "c:\my app\chara

# remote game data
dotnet charon DATA I18N IMPORT --dataBase "https://charon.live/view/data/My_Game/develop/" -
```

Parameters

--dataBase
Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials
The API key used to access remote server in case of
–*dataBase* being URL.

--schemas
A list of types of documents (schemas) to import. By default all schemas
EXCEPT metadata are imported.

- Use space to separate multiple schemas.

- You can use wildcards (*) at the beginning and end of names.

- You can use identifiers in {} instead of names.

- You can exclude certain names by using an exclamation mark (!) at the
  beginning of their names.

```
# schema name
--schemas Character
--schemas Character Item

# all (default)
--schemas *

# masks
--schemas Char*
--schemas *Modifier
--schemas *Mod*

# schema id
--schemas {18d4bf318f3c49688087dbed}

# negation
--schemas Char* !Character
--schemas !*Item*

# excluding system schemas (Schema, SchemaProperty, ProjectSetti
--schemas ![system]
```

--languages
The list of languages to import. Values are language
tags (BCP 47).

```
# Import specific language
--languages en-US es-ES

# Import all languages
--languages *
```

Overview

`--input`	Path to a file with data to import. Alternatively, you can use Standart Input or URL.

See input data structure requirements.

```
# standart input (default)
--input in
--input con

# absolute path (windows)
--input "c:\my app\input.json"

# absolute path (unix)
--input /user/data/input.json

# relative path (universal)
--input "./input.json"

# remote location (HTTP)
--input "http://example.com/input.json"

# remote location with authentication (FTP)
--input "ftp://user:password@example.com/input.json"
```

`--inputFormat`	Format of imported data.

```
# Auto-detect by extension (default)
--inputFormat auto

# XLIFF v2
--inputFormat xliff
--inputFormat xliff2

# XLIFF v1
--inputFormat xliff1

# XSLX Spreadsheet
--inputFormat xslx
```

`--inputFormattingOptions`	Additional options for specified format.

```
# absolute path (windows)
```

Overview

`--output`	Optional path to a import report file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output
--output out
--output con

# standart error
--output err

# null device (default)
--output null

# absolute path (windows)
--output "c:\my app\document.json"

# absolute path (unix)
--output /user/data/document.json

# relative path (universal)
--output "./document.json"

# remote location (HTTP)
--output "http://example.com/document.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/document.json"
```

`--outputFormat`	Format of import report.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XLSX Spreadsheet
--outputFormat xlsx
```

`--outputFormattingOptions`	Additional options for specified format.
`--dryRun`	Allows you to run the command without actually making any changes to the game data, providing a preview of what would happen.

This command supports universal parameters.

List Translation Languages

Get a list of supported translation languages. Primary language always shows up first in the list.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Overview

Command

```
# local game data (windows)
dotnet charon DATA I18N LANGUAGES --dataBase "c:\my app\gamedata.json" --output out --output

# remote game data
dotnet charon DATA I18N LANGUAGES --dataBase "https://charon.live/view/data/My_Game/develop/
```

Parameters

--dataBase   Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials   The API key used to access remote server in case of *–dataBase* being URL.

--output   Path to language list file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\input.json"

# absolute path (unix)
--output /user/data/input.json

# relative path (universal)
--output "./input.json"

# remote location (HTTP)
--output "http://example.com/input.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/input.json"
```

`--outputFormat`	Format of exported data.

```
# JSON (default)
--outputFormat json

#> [
#>    "en-US",
#>    "es-ES",
#> ]


# Space separated list
--outputFormat list

#> en-US es-ES

# New line (OS specific) separated list
--outputFormat table

#> en-US
   #> es-ES
```

`--outputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Import Data

Imports documents from file to a game data.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA IMPORT --dataBase "c:\my app\gamedata.json" --schemas Character --input "

# remote game data
dotnet charon DATA IMPORT --dataBase "https://charon.live/view/data/My_Game/develop/" --sche
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of –*dataBase* being URL.

`--schemas`	A list of types of documents (schemas) to import. By default all schemas *EXCEPT* metadata are imported.

- Use space to separate multiple schemas.

- You can use wildcards (*) at the beginning and end of names.

- You can use identifiers in {} instead of names.

- You can exclude certain names by using an exclamation mark (!) at the beginning of their names.

```
# schema name
--schemas Character
--schemas Character Item


# all (default)
--schemas *


# masks
--schemas Char*
--schemas *Modifier
--schemas *Mod*


# schema id
--schemas {18d4bf318f3c49688087dbed}


# negation
--schemas Char* !Character
--schemas !*Item*


# excluding system schemas (Schema, SchemaProperty, ProjectSettings)
--schemas ![system]
```

`--mode`	Import mode controls merge behavior during import.

```
# (default)
--mode createAndUpdate


--mode create
--mode update
--mode safeUpdate
--mode replace
--mode delete
```

createAndUpdate
    creates new documents and updates existing ones

create
    only creates new documents, existing documents are kept unchanged

update
    only updates existing documents, no new ones are created

safeUpdate
    same as *update* but without creating, moving and erasing embedded documents

replace
    replaces the entire collection with the imported documents

delete
    deletes documents found in the imported data

`--input`	Path to a data file. Alternatively, you can use Standart Input or URL.

```
# standart input (default)
--input in
--input con

# absolute path (windows)
--input "c:\my app\characters.json"

# absolute path (unix)
--input "/user/data/characters.json"

# relative path (universal)
--input "./characters.json"

# remote location (HTTP)
--input "http://example.com/characters.json"

# remote location with authentication (FTP)
--input "ftp://user:password@example.com/characters.json"
```

`--inputFormat`	Format of imported data.

```
# Auto-detect by extension (default)
--inputFormat auto

# JSON
--inputFormat json

# BSON
--inputFormat bson

# Message Pack
--inputFormat msgpack

# XML (removed in 2025.1.1)
--inputFormat xml

# XLSX Spreadsheet
--inputFormat xlsx
```

`--inputFormattingOptions`	Additional options for specified format.

Overview

`--output`	Optional path to a import report file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output
--output out
--output con

# standart error
--output err

# null device (default)
--output null

# absolute path (windows)
--output "c:\my app\document.json"

# absolute path (unix)
--output /user/data/document.json

# relative path (universal)
--output "./document.json"

# remote location (HTTP)
--output "http://example.com/document.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/document.json"
```

`--outputFormat`	Format of import report.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XLSX Spreadsheet
--outputFormat xlsx
```

`--outputFormattingOptions`	Additional options for specified format.
`--dryRun`	Allows you to run the command without actually making any changes to the game data, providing a preview of what would happen.

This command supports universal parameters.

Input Data Structure

The data you input should follow this structure (recommended):

```
{
  "Collections": {
    "<Schema-Name>": [
      {
        // <Document>
      }
    ]
  }
}
```

Overview

This structure is also accepted:

```
{
  "<Schema-Name>": [
    {
      // <Document>
    }
  ]
}
```

A list of documents is accepted if only one name in `--schemas` is specified:

```
[
  {
    // <Document>
  }
]
```

And single document is accepted too if only one name in `--schemas` is specified:

```
{
  // <Document>
}
```

List Documents

Seaches for a documents.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA LIST --dataBase "c:\my app\gamedata.json" --schema Character

# remote game data
dotnet charon DATA LIST --dataBase "https://charon.live/view/data/My_Game/develop/" --schema
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.
	
	`# local file`
	`--dataBase "c:\my app\gamedata.json"`
	
	`# remote server`
	`--dataBase "https://charon.live/view/data/My_Game/develop/"`

`--credentials`	The API key used to access remote server in case of *–dataBase* being URL.

`--schema`	Name or identifier of the type (schema) of document.
	
	`# name`
	`--schema Item`
	
	`# id`
	`--schema 55a4f32faca22e191098f3d9`

Overview

--filters   Document filter expressions.

```
# patterns
--filters <Field> <Operator> <Value> [<Field> <Operator> <Value>...]

# single expression
--filters Id > 10
--filters Name like "Zombie"

# multiple expressions
--filters Id > 10 Name like "Zombie"

# greater than
--filters Id > 0
--filters Id greaterThan 0


# greater than or equal
--filters Id >= 0
--filters Id greaterThanOrEqual 0


# less than
--filters Id < 0
--filters Id lessThan 0

# less than or equal
--filters Id <= 0
--filters Id LessThanOrEqual 0

# equal
--filters Id = 0
--filters Id == 0
--filters Id equal 0

# not equal
--filters Id <> 0
--filters Id != 0
--filters Id notEqual 0

# like - is used to search for specific patterns in a field, allowing for parti
--filters Name like "Zombie"
```

--sorters   Document sort expressions.

```
# patterns
--sorters <Field> ASC|DESC [<Field> ASC|DESC ...]

# ascending
--sorters Name ASC

# descending
--sorters Name DESC
```

Overview

`--path`	Embeddance path filter. Could be used to get only embedded documents.

```
# any path
--path *

# root documents (default)
--path ""

# in 'Item' property
--path /Item
```

`--skip`	Number of found documents to skip.

```
# skip first ten documents after applying --filter and --sort
--skip 10
```

`--take`	Max amount to documents return.

```
# limit to first 100 documents after --skip
--take 100
```

`--output`	Path to a found document file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output (default)
--output out
--output con

# standart error
--output err

# null device
--output null

# absolute path (windows)
--output "c:\my app\document.json"

# absolute path (unix)
--output /user/data/document.json

# relative path (universal)
--output "./document.json"

# remote location (HTTP)
--output "http://example.com/document.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/document.json"
```

`--outputFormat`	Format of exported data.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

`--outputFormattingOptions`	Additional options for specified format.

Overview

This command supports universal parameters.

Output

The exported data follows the general game data structure, but omits *ToolsVersion*, *RevisionHash*, and *ChangeNumber*.

```
{
  "Collections":
  {
    "Character":
    [
      {
        "Id": "Knight"

        /* rest of properties of document */
      },
      {
        "Id": "Templar"

        /* rest of properties of document */
      },
      // ...
    ]
  }
}
```

Restore from Backup

Restores game data from a file created by DATA BACKUP command.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA RESTORE --dataBase "c:\my app\gamedata.json" --input "c:\my app\backup.ms

# remote game data
dotnet charon DATA RESTORE --dataBase "https://charon.live/view/data/My_Game/develop/" --inp
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials	The API key used to access remote server in case of *–dataBase* being URL.

109

`--input`	Path to a backup file. Alternatively, you can use Standart Input or URL.

```
# standart input (default)
--input in
--input con

# absolute path (windows)
--input "c:\my app\backup.json"

# absolute path (unix)
--input "/user/data/backup.json"

# relative path (universal)
--input "./backup.json"

# remote location (HTTP)
--input "http://example.com/backup.json"

# remote location with authentication (FTP)
--input "ftp://user:password@example.com/backup.json"
```

`--inputFormat`	Format of imported data.

```
# Auto-detect by extension (default)
--inputFormat auto

# JSON
--inputFormat json

# Message Pack
--inputFormat msgpack
```

`--inputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Update Document

Updates a document. For a bulk updates use DATA IMPORT command with `--mode update`.
The update document in `--input` may be partial, with non-included fields being omitted.
Only the first document from the `--input` will be processed.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon DATA UPDATE --dataBase "c:\my app\gamedata.json" --schema Item --input "c:\my

# remote game data
dotnet charon DATA UPDATE --dataBase "https://charon.live/view/data/My_Game/develop/" --sche
```

Overview

Parameters

--dataBase

Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials

The API key used to access remote server in case of –*dataBase* being URL.

--schema

Name or identifier of the type (schema) of updated document.

```
# name
--schema Item

# id
--schema 55a4f32faca22e191098f3d9
```

--id

Identifier of updated document. Could be omitted if *Id* is specified in --input document.

```
# text
--id Sword

# number
--id 101
```

--input

Path to a file with update data. Alternatively, you can use Standart Input or URL.

```
# standart input (default)
--input in
--input con

# absolute path (windows)
--input "c:\my app\item.json"

# absolute path (unix)
--input "/user/data/item.json"

# relative path (universal)
--input "./item.json"

# remote location (HTTP)
--input "http://example.com/item.json"

# remote location with authentication (FTP)
--input "ftp://user:password@example.com/item.json"
```

Overview

`--inputFormat`	Format of update data.

```
# Auto-detect by extension (default)
--inputFormat auto

# JSON
--inputFormat json

# BSON
--inputFormat bson

# Message Pack
--inputFormat msgpack

# XML (removed in 2025.1.1)
--inputFormat xml
```

`--inputFormattingOptions`	Additional options for specified format.
`--output`	Path to a updated document file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output
--output out
--output con

# standart error
--output err

# null device (default)
--output null

# absolute path (windows)
--output "c:\my app\updated_item.json"

# absolute path (unix)
--output /user/data/updated_item.json

# relative path (universal)
--output "./updated_item.json"

# remote location (HTTP)
--output "http://example.com/updated_item.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/updated_item.json"
```

`--outputFormat`	Format of updated data.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

`--outputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Overview

Input Data Schema

The data you input should follow this schema (recommended):

```json
{
  "Collections": {
    "<Schema-Name>": [
      {
        // <Document>
      }
    ]
  }
}
```

This schema is also accepted:

```json
{
  "<Schema-Name>": [
    {
      // <Document>
    }
  ]
}
```

A list of documents is accepted:

```json
[
  {
    // <Document>
  }
]
```

And single document too:

```json
{
  // <Document>
}
```

Output

Outputs the updated document with all the edits that were made to make it conform to the schema.

```json
{
  "Id": "Sword"

  /* rest of properties of updated document */
}
```

Validate Game Data

Checks the game data for validity and produces a report.

The exit code will be `1` if the report contains errors and the `--output` is set to `err`. Otherwise, the exit code will be `0`.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Overview

```
# local game data (windows)
dotnet charon DATA DELETE --dataBase "c:\my app\gamedata.json" --schema Item --id "Sword"

# remote game data
dotnet charon DATA DELETE --dataBase "https://charon.live/view/data/My_Game/develop/" --sche
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials	The API key used to access remote server in case of –*dataBase* being URL.
--validationOptions	List of validation checks and repairs to perform.

```
# repairs
--validationOptions repair
--validationOptions deduplicateIds
--validationOptions repairRequiredWithDefaultValue
--validationOptions eraseInvalidValue

# checks (default)
--validationOptions checkTranslation
--validationOptions checkRequirements
--validationOptions checkFormat
--validationOptions checkUniqueness
--validationOptions checkReferences
--validationOptions checkSpecification
--validationOptions checkConstraints
```

Overview

`--output`	Path to a validation report file. If the file exists, it will be overwritten. The directory must already exist. Alternatively, you can output to Standard Error, Standard Output, /dev/null, or a URL.

```
# standart output
--output out
--output con

# standart error
--output err

# null device (default)
--output null

# absolute path (windows)
--output "c:\my app\document.json"

# absolute path (unix)
--output /user/data/document.json

# relative path (universal)
--output "./document.json"

# remote location (HTTP)
--output "http://example.com/document.json"

# remote location with authentication (FTP)
--output "ftp://user:password@example.com/document.json"
```

`--outputFormat`	Format of exported data.

```
# JSON (default)
--outputFormat json

# BSON
--outputFormat bson

# Message Pack
--outputFormat msgpack

# XML (removed in 2025.1.1)
--outputFormat xml
```

`--outputFormattingOptions`	Additional options for specified format.

This command supports universal parameters.

Output Data Schema

The report follow this pattern:

```
{
  records:
  [
    {
      id: "<document-id>",
      schemaId: "<schema-id>",
      schemaName: "<schema-name>",
      errors: // could be null if no errors
      [
        {
          path: "<path-in-document>",
          message: "<error-message>",
```

```
            code: "<error-code>"
         },
         // ...
      ]
    },
    // ...
  ]
}
```

or JSON schema:

```
{
    "type": "object",
    "x-name": "ValidationReport",
    "additionalProperties": false,
    "properties": {
      "records": {
        "type": "array",
        "items": {
          "type": "object",
          "x-name": "ValidationRecord",
          "additionalProperties": false,
          "properties": {
            "id": { },
            "schemaName": {
              "type": "string"
            },
            "schemaId": {
              "type": "string"
            },
            "errors": {
              "type": "array",
              "items": {
                "type": "object",
                "x-name": "ValidationError",
                "additionalProperties": true,
                "readOnly": true,
                "properties": {
                  "path": {
                    "type": "string"
                  },
                  "message": {
                    "type": "string"
                  },
                  "code": {
                    "type": "string"
                  }
                }
              }
            }
          }
        }
      },
      "metadataHashCode": {
        "type": "integer",
        "format": "int32"
      }
    }
}
```

Generate C# Source Code

Generates C# source code for game data into output directory.

This command does not delete previously generated files, and it is the responsibility of the user to ensure that any previous files are removed before running the command again.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon GENERATE CSHARPCODE --dataBase "c:\my app\gamedata.json" --namespace "MyGame.P

# remote game data
dotnet charon GENERATE CSHARPCODE --dataBase "https://charon.live/view/data/My_Game/develop/
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of *–dataBase* being URL.
`--outputDirectory`	Specifies the path where the source code should be written. It can be either an absolute or relative path to a directory. The specified directory must already be present.

```
# Windows
--outputDirectory "c:\my app\scripts"

# Linux or OSX
--outputDirectory "~/my app/scripts"

# Relative path
--outputDirectory "./my app/scripts"
```

`--languageVersion`	Target C# version. By default it is 4.0.

```
--languageVersion CSharp40
--languageVersion CSharp73
```

`--documentClassName`	Name for base class for all documents.

```
# name (default)
--documentClassName Document

# in case of name collision
--documentClassName GameDataDocument
```

Overview

`--gameDataClassName`	Name for class containing whole in-memory game data.

```
# name (default)
--gameDataClassName GameData

# in case of name collision
--gameDataClassName MyGameData
```

`--namespace`	Namespace for all generated classes.

```
# name (default)
--namespace GameParameters
```

`--defineConstants`	Preprocessor constants to define. Use semicolon(;) to separate multiple values.

```
# Use GameDevWare.Dynamic.Expressions.dll for formulas
--defineConstants USE_DYNAMIC_EXPRESSIONS

# Exclude all formula related code from compilation
--defineConstants SUPPRESS_BUILD_IN_FORMULAS

# Enable some JSON formatting optimizations using System.Memory.dll and Sy
--defineConstants BUFFERS_TEXT_DLL
```

`--indentation`	Indentation style for generated code.

```
# Tabs (default)
--indentation Tabs

# Two spaces
--indentation TwoSpaces

# Four spaces
--indentation FourSpaces
```

`--lineEndings`	Line ending symbols for generated code.

```
# Windows \\r\\n (default)
--lineEndings Windows

# Unix style \\n
--lineEndings Unix
```

`--splitFiles`	Set this flag to lay out generated classes into separate files. If not set, then one giant file with the name of `--gameDataClassName`.cs will be generated.

`--optimizations` list of enabled optimization in generated code.

```
# Eagerly resolves and validates all references in loaded documents.
# When enabled, this optimization ensures that all references in documents are
# during loading. This comes with a performance cost but guarantees the validit
--optimizations eagerReferenceResolution

# Opts for raw references without generating helper methods for referenced docu
# With this optimization, the generated code will not include helper methods fo
# referenced documents, keeping only accessors that work with raw references.
--optimizations rawReferences

# Avoids generating helper methods for localized strings, keeping only raw acce
# This optimization eliminates helper methods for accessing localized text, ins
# accessors that deal directly with lists of localized texts.
--optimizations rawLocalizedStrings

# Disables string pooling during game data loading.
# Turning off string pooling can yield a minor performance improvement at the c
# memory usage, as it avoids reusing short strings.
--optimizations disableStringPooling

# Disables generation of code for loading game data from JSON formatted files.
# This optimization omits code related to JSON serialization, useful when JSON
# game data is not used.
--optimizations disableJsonSerialization

# Disables generation of code for loading game data from Message Pack formatted
# Similar to DisableJsonSerialization, this option removes code related to load
# from Message Pack formatted files.
--optimizations disableMessagePackSerialization

# Disables generation of code related to applying patches during game data load
# This removes a significant portion of code that is mainly used for modding su
# where patches are applied to game data at runtime.
--optimizations disablePatching

# Disables generation of enums with known document IDs.
# This removes a significant portion of code that contains listings of IDs for
# documents known at the moment of code generation, which improves compilation
--optimizations disableDocumentIdEnums
```

`--clearOutputDirectory`   Clear the output directory from generated files. Generated files are identified by the presence of the '<auto-generated>' tag inside.

This command supports universal parameters.

Generate Haxe Source Code

Generates Haxe source code for game data into output directory.

This command does not delete previously generated files, and it is the responsibility of the user to ensure that any previous files are removed before running the command again.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Overview

```
# local game data (windows)
dotnet charon GENERATE HAXE --dataBase "c:\my app\gamedata.json" --packageName "" --outputDi

# remote game data
dotnet charon GENERATE HAXE --dataBase "https://charon.live/view/data/My_Game/develop/" --pa
```

Parameters

--dataBase	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials	The API key used to access remote server in case of *–dataBase* being URL.
--outputDirectory	Specifies the path where the source code should be written. It can be either an absolute or relative path to a directory. The specified directory must already be present.

```
# Windows
--outputDirectory "c:\my app\scripts"

# Linux or OSX
--outputDirectory "~/my app/scripts"

# Relative path
--outputDirectory "./my app/scripts"
```

--documentClassName	Name for base class for all documents.

```
# name (default)
--documentClassName Document

# in case of name collision
--documentClassName GameDataDocument
```

--gameDataClassName	Name for class containing whole in-memory game data.

```
# name (default)
--gameDataClassName GameData

# in case of name collision
--gameDataClassName MyGameData
```

--packageName	Package name for all generated classes.

```
# empty package (default)
--packageName ""

# named
--packageName GameParameters
```

Overview

`--indentation`		Indentation style for generated code.

```
# Tabs (default)
--indentation Tabs

# Two spaces
--indentation TwoSpaces

# Four spaces
--indentation FourSpaces
```

`--lineEndings`	Line ending symbols for generated code.

```
# Windows \\r\\n (default)
--lineEndings Windows

# Unix style \\n
--lineEndings Unix
```

`--splitFiles`	Set this flag to lay out generated classes into separate files. If not set, then one giant file with the name of `--gameDataClassName`.hx will be generated.

`--optimizations`	List of enabled optimization in generated code.

```
# Eagerly resolves and validates all references in loaded documents.
# When enabled, this optimization ensures that all references in documents
# during loading. This comes with a performance cost but guarantees the va
--optimizations eagerReferenceResolution

# Opts for raw references without generating helper methods for referenced
# With this optimization, the generated code will not include helper metho
# referenced documents, keeping only accessors that work with raw referenc
--optimizations rawReferences

# Avoids generating helper methods for localized strings, keeping only raw
# This optimization eliminates helper methods for accessing localized text
# accessors that deal directly with lists of localized texts.
--optimizations rawLocalizedStrings

# Disables string pooling during game data loading.
# Turning off string pooling can yield a minor performance improvement at
# memory usage, as it avoids reusing short strings.
--optimizations disableStringPooling

# Disables generation of code for loading game data from JSON formatted fi
# This optimization omits code related to JSON serialization, useful when
# game data is not used.
--optimizations disableJsonSerialization

# Disables generation of code for loading game data from Message Pack form
# Similar to DisableJsonSerialization, this option removes code related to
# from Message Pack formatted files.
--optimizations disableMessagePackSerialization

# Disables generation of code related to applying patches during game data
# This removes a significant portion of code that is mainly used for moddi
# where patches are applied to game data at runtime.
--optimizations disablePatching

# Disables generation of enums with known document IDs.
# This removes a significant portion of code that contains listings of IDs
# documents known at the moment of code generation, which improves compila
--optimizations disableDocumentIdEnums
```

| `--clearOutputDirectory` | Clear the output directory from generated files. Generated files are identified by the presence of the '<auto-generated>' tag inside. |

This command supports universal parameters.

Export Code Generation Templates

Exports T4 code generation templates to a specified directory. These templates can be used with Visual Studio, Rider, Visual Studio Code with plugin, `dotnet` tool or other tools to generate source code.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# Windows
dotnet charon GENERATE TEMPLATES --outputDirectory "c:\templates"
```

Parameters

| `--outputDirectory` | Specifies the path where the templates should be written. It can be either an absolute or relative path to a directory. The specified directory must already be present. |

```
# Windows
--outputDirectory "c:\templates"

# Linux or OSX
--outputDirectory "~/templates"

# Relative path
--outputDirectory "./templates"
```

Generate Text from Templates (Obsolete)

Generates text from T4 templates into output directory.

> **Warning**
>
> This command was removed since 2025.1.1 version. It is recommended to use an IDE or open-source alternatives for generating text with T4 templates. See: dotnet-t4

See GENERATE TEMPLATES to get actual T4 templates.

- CLI Installation
- Commands Reference

Generate TypeScript Source Code

Generates TypeScript source code for game data into output directory.

This command does not delete previously generated files, and it is the responsibility of the user to ensure that any previous files are removed before running the command again.

- CLI Installation

Overview

- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon GENERATE TYPESCRIPTCODE --dataBase "c:\my app\gamedata.json" --outputDirectory

# remote game data
dotnet charon GENERATE TYPESCRIPTCODE --dataBase "https://charon.live/view/data/My_Game/deve
```

Parameters

--dataBase
Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\my app\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

--credentials
The API key used to access remote server in case of *–dataBase* being URL.

--outputDirectory
Specifies the path where the source code should be written. It can be either an absolute or relative path to a directory. The specified directory must already be present.

```
# Windows
--outputDirectory "c:\my app\scripts"

# Linux or OSX
--outputDirectory "~/my app/scripts"

# Relative path
--outputDirectory "./my app/scripts"
```

--documentClassName
Name for base class for all documents.

```
# name (default)
--documentClassName Document

# in case of name collision
--documentClassName GameDataDocument
```

--gameDataClassName
Name for class containing whole in-memory game data.

```
# name (default)
--gameDataClassName GameData

# in case of name collision
--gameDataClassName MyGameData
```

Overview

`--indentation`	Indentation style for generated code.

```
# Tabs (default)
--indentation Tabs

# Two spaces
--indentation TwoSpaces

# Four spaces
--indentation FourSpaces
```

`--lineEndings`	Line ending symbols for generated code.

```
# Windows \\r\\n (default)
--lineEndings Windows

# Unix style \\n
--lineEndings Unix
```

`--splitFiles`	Set this flag to lay out generated classes into separate files. If not set, then one giant file with the name of `--gameDataClassName`.ts will be generated.
`--optimizations`	List of enabled optimization in generated code.

```
# Eagerly resolves and validates all references in loaded documents.
# When enabled, this optimization ensures that all references in documents
# during loading. This comes with a performance cost but guarantees the va
--optimizations eagerReferenceResolution

# Opts for raw references without generating helper methods for referenced
# With this optimization, the generated code will not include helper metho
# referenced documents, keeping only accessors that work with raw referenc
--optimizations rawReferences

# Avoids generating helper methods for localized strings, keeping only raw
# This optimization eliminates helper methods for accessing localized text
# accessors that deal directly with lists of localized texts.
--optimizations rawLocalizedStrings

# Disables string pooling during game data loading.
# Turning off string pooling can yield a minor performance improvement at
# memory usage, as it avoids reusing short strings.
--optimizations disableStringPooling

# Disables generation of code for loading game data from JSON formatted fi
# This optimization omits code related to JSON serialization, useful when
# game data is not used.
--optimizations disableJsonSerialization

# Disables generation of code for loading game data from Message Pack form
# Similar to DisableJsonSerialization, this option removes code related to
# from Message Pack formatted files.
--optimizations disableMessagePackSerialization

# Disables generation of code related to applying patches during game data
# This removes a significant portion of code that is mainly used for moddi
# where patches are applied to game data at runtime.
--optimizations disablePatching

# Disables generation of enums with known document IDs.
# This removes a significant portion of code that contains listings of IDs
# documents known at the moment of code generation, which improves compila
--optimizations disableDocumentIdEnums
```

`--clearOutputDirectory`	Clear the output directory from generated files. Generated files are identified by the presence of the '<auto-generated>' tag inside.

This command supports universal parameters.

Generate Unreal Engine C++ Source Code

Generates C++ for Unreal Engine source code for game data into output directory.

This command does not delete previously generated files, and it is the responsibility of the user to ensure that any previous files are removed before running the command again.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon GENERATE UECPPCODE --dataBase "c:\My Project\Content\gamedata.json" --outputDi

# remote game data
dotnet charon GENERATE UECPPCODE --dataBase "https://charon.live/view/data/My_Game/develop/"
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.

```
# local file
--dataBase "c:\My Project\Content\gamedata.json"

# remote server
--dataBase "https://charon.live/view/data/My_Game/develop/"
```

`--credentials`	The API key used to access remote server in case of *–dataBase* being URL.
`--outputDirectory`	Specifies the path where the source code should be written. It can be either an absolute or relative path to a directory. The specified directory must already be present.

```
# Windows
--outputDirectory "c:\My Project\Source\Gamedata"

# Linux or OSX
--outputDirectory "~/My Project/Source/Gamedata"

# Relative path
--outputDirectory "./My Project/Source/Gamedata"
```

`--documentClassName`	Name for base class for all documents.

```
# name (default)
--documentClassName Document # became UDocument in generated code

# in case of custom inheritance chain
    # class SHOULD publicly inherit UDocument
--documentClassName GameDataDocument # became UGameDataDocument in ge
```

Overview

`--gameDataClassName` Name for class containing whole in-memory game data.

```
# name (default)
--gameDataClassName GameData # became UGameData in generated code

# in case of name collision
--gameDataClassName MyGameData # became UMyGameData in generated code
```

`--defineConstants` Preprocessor constants to define. Use semicolon(;) to separate multiple values.

```
--defineConstants NO_OPTIMIZATIONS;USE_FSTRING_ONLY
```

`--indentation` Indentation style for generated code.

```
# Tabs (default)
--indentation Tabs

# Two spaces
--indentation TwoSpaces

# Four spaces
--indentation FourSpaces
```

`--lineEndings` Line ending symbols for generated code.

```
# Windows \\r\\n (default)
--lineEndings Windows

# Unix style \\n
--lineEndings Unix
```

`--optimizations` list of enabled optimization in generated code.

```
# Eagerly resolves and validates all references in loaded documents.
# When enabled, this optimization ensures that all references in documents are
# during loading. This comes with a performance cost but guarantees the validit
--optimizations eagerReferenceResolution

# Opts for raw references without generating helper methods for referenced docu
# With this optimization, the generated code will not include helper methods fo
# referenced documents, keeping only accessors that work with raw references.
--optimizations rawReferences

# Avoids generating helper methods for localized strings, keeping only raw acce
# This optimization eliminates helper methods for accessing localized text, ins
# accessors that deal directly with lists of localized texts.
--optimizations rawLocalizedStrings

# Disables string pooling during game data loading.
# Turning off string pooling can yield a minor performance improvement at the c
# memory usage, as it avoids reusing short strings.
--optimizations disableStringPooling

# Disables generation of code for loading game data from JSON formatted files.
# This optimization omits code related to JSON serialization, useful when JSON
# game data is not used.
--optimizations disableJsonSerialization

# Disables generation of code for loading game data from Message Pack formatted
# Similar to DisableJsonSerialization, this option removes code related to load
# from Message Pack formatted files.
--optimizations disableMessagePackSerialization

# Disables generation of code related to applying patches during game data load
# This removes a significant portion of code that is mainly used for modding su
# where patches are applied to game data at runtime.
--optimizations disablePatching

# Disables generation of enums with known document IDs.
# This removes a significant portion of code that contains listings of IDs for
# documents known at the moment of code generation, which improves compilation
--optimizations disableDocumentIdEnums
```

`--clearOutputDirectory` Clear the output directory from generated files. Generated files are identified by the presence of the '<auto-generated>' tag inside.

This command supports universal parameters.

Initialize Game Data

Initializes an empty or missing file with initial data. Path to game data should be local file system's file.

- CLI Installation
- Commands Reference

Overview

Command

```
# full path (windows)
dotnet charon INIT "c:\my app\gamedata.gdjs"

# full path (linux)
dotnet charon INIT "/var/mygame/gamedata.json"

# relative path
dotnet charon INIT mygame/gamedata.json
```

Parameters

--fileName

Absolute or relative path to game data file. Use quotation marks if your path contains spaces.

```
# local file
--fileName "c:\my app\gamedata.json"
```

URL input/output parameters

Some command accept URL as input/output parameter.

Supported URL Schemes

Scheme	Input parameter	Output parameter
HTTP[S]	A GET request will be sent	A POST request with body containing output will be sent
FTP(S)	A RETR command will be sent	A STOR command with output content will be sent
FILE	File will be read	File will be written

Authentication

Authentication data could be passed in *user* part of URL. More advanced authentication schemes are not supported.

relative path

Overview

```
# publish data to FTP
dotnet charon DATA EXPORT
   --dataBase "https://charon.live/view/data/My_Game/develop/dashboard"
   --output "ftp://user:password@example.com/public/gamedata.json"
   --mode publication
   --outputFormat json
   --credentials "<API-Key>"

# import localization from remote HTTP server
dotnet charon DATA I18N IMPORT
   --dataBase "file:///c:/my app/gamedata.json"
   --input "https://example.com/translated/gamedata.xliff"
   --inputFormat xliff

# print languages for game data in local file
dotnet charon DATA I18N LANGUAGES --dataBase "file:///c:/my app/gamedata.json"

# print languages for game data in local file relative to current working directory
dotnet charon DATA I18N LANGUAGES --dataBase "file:///./gamedata.json"

# print languages for game data at remote server using API Key
export CHARON_API_KEY=87758CC0D7C745D0948F2A8AFE61BC81
dotnet charon DATA I18N LANGUAGES --dataBase "https://charon.live/view/data/My_Game/develop/
```

Start in Standalone Mode

Starts Charon in standalone mode for specified game data. Path to game data could be local file system's file or remote server address.

- CLI Installation
- Commands Reference
- Universal Parameters
- URL-based Input/Output

Command

```
# local game data (windows)
dotnet charon SERVER START --dataBase "c:\my app\gamedata.json" --port 8080 --launchDefaultB

# shortcut version
dotnet charon "c:\my app\gamedata.json"
```

Parameters

`--dataBase`	Absolute or relative path to game data. Use quotation marks if your path contains spaces.
	`# local file` `--dataBase "c:\my app\gamedata.json"`
`--port`	Number of an IP port (1-65535) to be used to listen for browser based UI.
`--launchDefaultBrowser`	Set this flag to open system-default browser on successful start.
`--resetPreferences`	Set this flag to reset UI preferences on successful start.

This command supports universal parameters.

Universal parameters

All commands accept universal parameters and environment variables.

`--verbose`	Set this flag to get additional diagnostic information in logs.
`--log <path>`	Add additional file logging to the existing logging configuration from `appsettings.json`.

```
--log "./logs/charon.log"
```

`--log out`	Add additional terminal (standard output) logging to the existing logging configuration from `appsettings.json`.

```
--log out
# or
--log con
```

`--pause`	Wait for user prompt before the application exits.

Environment variables

In addition to the standard configuration redefinition mechanism using environment variables, the following environment variables are also supported.

CHARON_API_KEY

The **API key** which is used to access the remote server. This environment variable is usually used in conjunction with `--dataBase`, which points to a remote server.

```
# Windows
set CHARON_API_KEY=87758CC0D7C745D0948F2A8AFE61BC81

# OSX or Linux
export CHARON_API_KEY=87758CC0D7C745D0948F2A8AFE61BC81
```

Get Charon Version

Gets version of `dotnet charon` application.
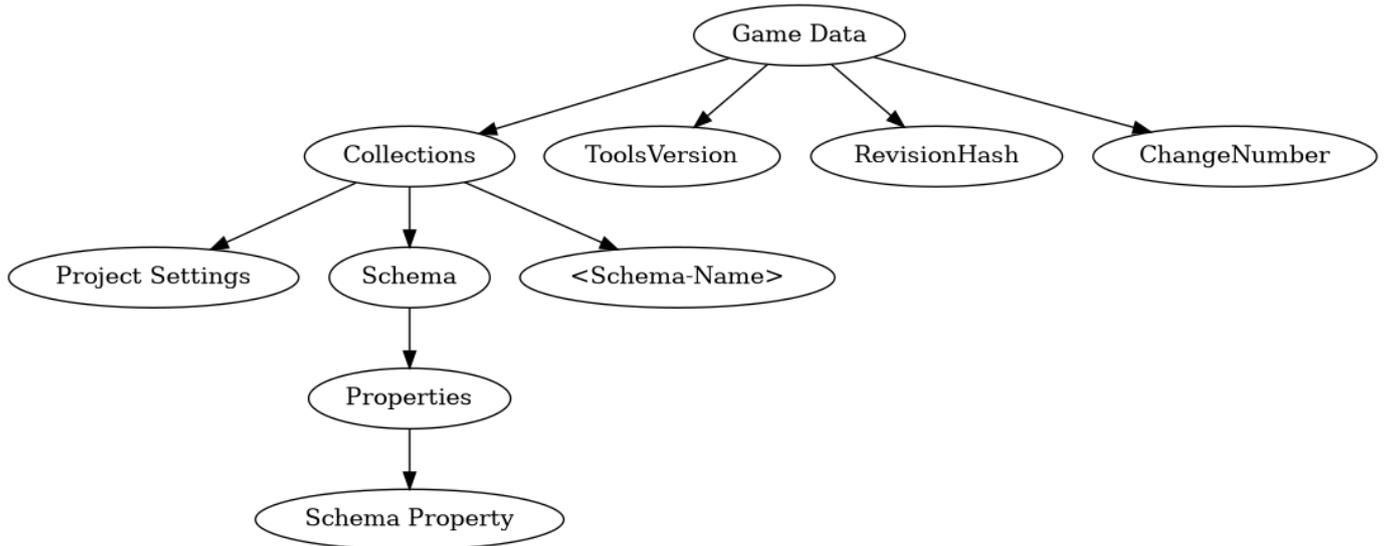
- CLI Installation
- Commands Reference

Command

```
# Windows, Linux or OSX
dotnet charon VERSION
#> 2023.2.3-alpha
```

Parameters

This command supports universal parameters.

Game Data Structure

Game Data

Fields

- `ToolsVersion` (string): Version of the application used to create this file.
- `RevisionHash` (string): Current changeset hash value.
- `ChangeNumber` (number): Current changeset ordinal number.
- `Collections` (object): List of document collections identified by schema name.

  - `ProjectSettings` (array): Project-related settings for the current file.

    - See Project Settings section below.
  - `Schema` (array): Project-related schemas for the current file.

    - See Schema section below.
  - `<Schema-Name>` (array): Other document collections listed in alphabetical order.

Example

```json
{
  "ToolsVersion": "2023.1.2.0",
  "RevisionHash": "678f998993a22d1f54b7fa80",
  "ChangeNumber": 1,
  "Collections":
  {
    "ProjectSettings":
    [
      {
        /* see project settings section below */
      }
    ],
    "Schema":
    [
      {
        /* see schema section below */
      }
    ],
    "SchemaProperty": [ /* always empty */ ],
```

Overview

```
"<Schema-Name>":
[
  {
    "Id": "<Id>" // All documents have Id

    /* rest of properties of document */
  },
  // ...
]
}
}
```

Project Settings

Fields

- `Id` (string): Unique identifier for the project settings (BSON ObjectId).

- `Name` (string): Name of the project.

- `PrimaryLanguage` (string): Primary language for localizable text in the project (language ID in BCP-47 format).

- `Languages` (string): Alternative languages for localizable text in the project (semicolon-delimited list of language IDs in BCP-47 format).

- `Copyright` (string): Copyright information for the project.

- `Version` (string): Version of the current file, represented as four numbers separated by dots (Major.Minor.Build.Revision).

Example

```
{
  "Id": "049bc0604c363a980b000088",
  "Name": "My Project",
  "PrimaryLanguage": "en-US",
  "Languages": "en-GB;fr-FR",
  "Copyright": "My Company (■) 2025",
  "Version": "1.0.0.0"
}
```

Schema

Fields

- `Id` (string): Unique identifier for the schema (BSON ObjectId).

- `Name` (string): Name of the schema (valid C identifier).

- `DisplayName` (string): Display name of the schema for UI purposes.

- `Description` (string): Schema description used in generated documentation.

- `Specification` (string): Extension data for the schema in *application/x-www-form-urlencoded* format (RFC-1867).

- `IdGenerator` (number): ID generation method for documents created by this schema:

    - *0* : None - ID must be provided manually by the user.

    - *1* : ObjectId - Generates a new BSON ObjectId.

    - *2* : Guid - Generates a new UUID.

    - *3* : Sequence - Uses an incrementing number unique to each schema.

    - *4* : GlobalSequence - Uses an incrementing number shared across all schemas.

Overview

- `Type` (number): Schema type:

    - *0* : Normal - Documents can be created in *Collections* or embedded in another document.
    - *1* : Component - Documents are always embedded in another document and never appear in `Collections`.
    - *2* : Settings - Only one document of this schema can exist in `Collections`.
- `Properties` (array): List of schema properties. Always includes the `Id` property.

    - See Schema Property section below.

Example

```
{
  "Id": "592fc86c983a36266c0912a0",
  "Name": "Item",
  "DisplayName": "Items",
  "Type": 0,
  "Description": "An item.",
  "IdGenerator": 1,
  "Specification": "icon=fugue16%2Fabacus&group=Metagame",
  "Properties": [
    // property
  ]
}
```

Schema Property

Structure

- `Id` (string): Unique identifier for the property (BSON ObjectId).
- `Name` (string): Name of the property (valid C identifier).
- `DisplayName` (string): Display name for UI and documentation purposes.
- `Description` (string): Property description used in generated documentation.
- `DataType` (number): Data type of values stored in documents:

    - *0*: Text - Line of text.
    - *1*: LocalizedText - Lines of localized text.
    - *5*: Logical - Boolean value.
    - *8*: Time - Time span.
    - *9*: Date - Specific date.
    - *12* : Number - Decimal number.
    - *13* : Integer - Whole number.
    - *18* : PickList - Predefined value list.
    - *19* : MultiPickList - Multiple selections from predefined values.
    - *22* : Document - Embedded document.
    - *23* : DocumentCollection - Collection of embedded documents.
    - *28* : Reference - Reference to another document.
    - *29* : ReferenceCollection - References to multiple documents.
    - *35* : Formula - C#-like expression for calculations.
- `DefaultValue` (vary|null): Default value for the property used when a new document is created.

Overview

- `Uniqueness` (number): Uniqueness requirement for the property:

  - *0* : None - Value does not need to be unique.

  - *1* : Unique - Value must be unique across all documents of this type.

  - *2* : UniqueInCollection - Value must be unique within the containing collection.

- `Requirement` (number): Value requirement for the property:

  - *0* : None - Value is optional and can be null.

  - *2* : NotNull - Value is required but can be an empty string/collection.

  - *3* : NotEmpty - Value is required and cannot be empty.

- `ReferenceType` (object|null): Referenced schema for certain data types (*Document*, *DocumentCollection*, *Reference*, *ReferenceCollection*):

  - `Id` (string): Identifier of the referenced schema.

  - `DisplayName` (string): Optional display name of the referenced schema.

- `Size` (number): Maximum or exact size of the data type. For variable-length types (e.g., text, collections), this defines the size; for others, it is zero.

- `Specification` (string): Extension data for the property in *application/x-www-form-urlencoded* format (RFC-1867).

Example

```
{
  "Id": "592fc9f8983a36266c0912aa",
  "Name": "TextField",
  "DisplayName": "Text Field",
  "Description": "",
  "DataType": 0,
  "DefaultValue": null,
  "Uniqueness": 0,
  "Requirement": 0,
  "ReferenceType": null,
  "Size": 0,
  "Specification": null
}
```

Internationalization (i18n)

Charon supports storing text data in multiple languages by using the special `LocalizedText` data type.

A list of possible translation languages is defined in the `Project Settings`.

There are two ways to pass translatable text to a third party (e.g., for localization or editing):

- You can export all translatable data as an XLSX spreadsheet.

- You can use the special localization format, XLIFF (XML Localization Interchange File Format).

Translation flow via UI

To export and translate your project's data, follow these steps:

1. Go to the dashboard of the project for which you want to generate source code.

2. Click on the "Internationalization Settings" link.

3. Click on the "Export" button to export the data.

4. Download the exported file and provide it to your translation team for translation.

5. Once the data has been translated, click on the "Import" button on the same page.

6. Select the translated file and follow the steps provided in the import wizard.

Translation flow via CLI

Exporting to XLSX spreadsheet

To export translatable text data as *XLSX*, run the DATA EXPORT command with the following parameters:

```
dotnet charon DATA EXPORT --dataBase "c:\my app\gamedata.json" --properties [LocalizedText]
```

- Use `--properties [LocalizedText]` parameter to indicate that only the properties containing `LocalizedText` should be exported.
- Use `--languages` parameter to limit the number of exported languages.

Extra columns may be present in the export files, which are required for the correct import of the translated data.

Importing from XLSX spreadsheet

Once your data is processed (e.g., translated), you can import it using the DATA IMPORT command with the `safeUpdate` mode:

```
dotnet charon DATA IMPORT --dataBase "c:\my app\gamedata.json" --input "c:\my app\text_all_l
```

Exporting to XLIFF

To export translatable text data as *XLIFF*, run the DATA I18N EXPORT command with the following parameters:

```
dotnet charon DATA I18N EXPORT --dataBase "c:\my app\gamedata.json" --sourceLanguage en --ta
```

- Use the `--outputFormat` parameter to indicate the exact format of the exported data, which can be either *xliff*, *xiff1*, or *xliff2*.
- Use `--sourceLanguage` to indicate the language text is being translated from as the *source*, and `--targetLanguage` to indicate the *target* language that the text is being translated to.
- To get a list of configured translation languages for the game data, run the DATA I18N LANGUAGES command.

Importing from XLIFF

Once the data has been processed, you can import it using the DATA I18N IMPORT command.

```
dotnet charon DATA I18N IMPORT --dataBase "c:\my app\gamedata.json" --input "c:\my app\en_fr
```

Other formats

While the export and import commands may accept other formats, it cannot be guaranteed that they will be supported.

Working with Logs

Charon creates a log files with various messages that may be useful for troubleshooting and debugging.

Unity Plugin

Log files are saved to `<project-directory>/Library/Charon/logs/`.

Unreal Engine Plugin

Log files are saved to `<project-directory>/Intermediate/Charon/logs/`.

CLI and Standalone

Log files are saved to:

- Windows: ``C:/Users/%USERNAME%/AppData/Roaming/Charon/logs/`.

- MacOS: `~/Library/Application Support/Charon/logs`

- Linux: `~/.config/Charon/logs`

Note: Make sure to replace `<project-directory>` and `<charon-directory>` with the actual directories on your system.

Logging Levels

Normally only the most important events are logged. If you have trouble identifying an issue, you might want to change log to *verbose*. This way more information is included in logs.

Unity Plugin

In menu select `Tools → Charon → Troubleshooting → Verbose Logs`.

CLI and Standalone

Launch with `--verbose` parameter.

Then repeat the action that causes the bug (or the one you want analyzed anyway) and check log file again.

CLI Example:

```
dotnet charon SERVER START ./gamedata.json --launchDefaultBrowser --verbose
```

Resetting UI Preferences

If for some reason editor behaves erratically (grids aren't displayed correctly or aren't displayed at all), you can restore default UI settings.

Unity plugin

Select in menu `Tools → Charon → Troubleshooting → Reset Preferences`.

CLI and Standalone

Launch with `--resetPreferences.` parameter.

Web

Use the `Preferences` profile tab `<User Icon> → Profile → Preferences`.

CLI Example:

```
dotnet charon SERVER START ./gamedata.json --launchDefaultBrowser --resetPreferences
```

Frequently Asked Questions (FAQ)

Is schema inheritance available?

Yes, "inheritance" is available in Charon. However, instead of traditional inheritance, Charon adopts a composition-based approach for extending schemas. This means that you can enhance and expand the functionality of existing schemas by including them into another schemas as `Document` properties. Read more about it.

Glossary

Game Data

The static information for the game, such as items, quests, dialogues, etc., is stored as game data. Schemas are also included to organize and structure game data.

Schema

A schema is a description of the structure for documents in game data. It defines the properties and structure of each document in the game data.

Schema Property

A part of a schema that defines a specific property or attribute of a document in the game data.

Formula

A property data type in a schema for a C# expression that can be executed at runtime to calculate a value for a field.

Reference

A property data type in a schema for a pointer to another document.

Document

A specific instance of a schema in the game data. It represents a single item or entity in the game, such as an item, quest, or dialogue.

Field

A named part of a document that holds a specific value.

Source Code

The code generated by Charon that represents the game data. This code can be used to load the game data at runtime.

Metadata

All the schemas and relations between them. This data is used by Charon to generate the source code for the game data.

Workspace

In the web application, the workspace is the place where users can manage their projects and subscription.

Project

In the web application, a project is a container for organizing related game data. It can contain multiple branches.

Branch

In the web application, a branch is a specific variant of game data within a project. It can be used to manage different versions or stages of the game data.

API Key

A unique identifier generated for a user in the User's Profile "API Keys" section, which can be used to access the REST API and CLI for various operations. It allows for automation of game build processes, such as pushing game data to local GIT repositories.

Publication

The process of exporting game data in a format that can be loaded into the game.

HTTP Routing Table

/app

PUT /app/log/

Log specified message on server. Used internally while standalone-hosted.

/auth

POST /auth/flow/api-key/

null

POST /auth/flow/email-code/

null

POST /auth/flow/oauth2/{authenticationProvider}/complete/

null

POST /auth/flow/oauth2/{authenticationProvider}/prepare/

null

POST /auth/flow/on-behalf/

null

POST /auth/flow/password/

null

POST /auth/one-time-code/

null

/billing

GET /billing/{userId}/account/

Get billing account by id.

GET /billing/{userId}/payment/status/

Get status of payment for subscription for workspace.

POST /billing/notification/

Accept notification from payment gate.

POST /billing/{userId}/account/

Update billing information

POST /billing/{userId}/contact-request/

Request contact from sales representative.

POST /billing/{userId}/customer-portal/

Get url of customer portal for user if available.

POST /billing/{userId}/payment/

Make payment for selected invoice.

POST /billing/{userId}/payment/status/

Start subscription session for workspace.

POST /billing/{userId}/payment/upcoming/

Get prorated upcoming payment information.

/context

GET /context/

Get page context.

/datasource

GET /datasource/{dataSourceId}/

Backup data source.

GET /datasource/{dataSourceId}/capabilities/

Get data source's capabilities.

GET /datasource/{dataSourceId}/collection/{schemaNameOrId}/

Find document by it's id or unique property value.

GET /datasource/{dataSourceId}/collections/

Export documents from multiple collections.

GET /datasource/{dataSourceId}/collections/raw/

Export documents from multiple collections into downloadable format without response wrapper.

GET /datasource/{dataSourceId}/documents/query/

Query documents from all collections.

GET /datasource/{dataSourceId}/formula/type/

List formula types.

GET /datasource/{dataSourceId}/loading-progress/

Get data source's loading progress.

GET /datasource/{dataSourceId}/present-users/

Get list of users present in specificated data source.

GET /datasource/{dataSourceId}/process/

List processes.

GET /datasource/{dataSourceId}/process/{processId}/

Get process's state.

GET /datasource/{dataSourceId}/process/{processId}/result/raw/

Get process's execution result without response wrapper.

GET /datasource/{dataSourceId}/raw/

Backup data source into downloadable format without response wrapper.

GET /datasource/{dataSourceId}/source-code/templates/

Get T4 templates for generating source code.

GET /datasource/{dataSourceId}/stats/

Get data source's statistics.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/

Update document.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/documents/

List documents.

POST /datasource/{dataSourceId}/collection/{schemaNameOrId}/translation/

Machine translate specified document. First language in the list is a source language.

POST /datasource/{dataSourceId}/completion/schema/

Suggest schema structure with specified AI tool.

POST /datasource/{dataSourceId}/completion/schema/icon/

Suggest an icon for schema using AI tool.

POST /datasource/{dataSourceId}/completion/thread/{threadId}/

Send AI chat message to specified chat thread.

POST /datasource/{dataSourceId}/converter/raw/

Convert specified game data documents from request body to JSON format and return it without response wrapper.

POST /datasource/{dataSourceId}/documents/query/

Pick multiple documents by their unique properties e.g. batched find request. Max documents per request is - 20.

POST /datasource/{dataSourceId}/process/{processId}/

Stop process.

POST /datasource/{dataSourceId}/source-code/

Generate source code for data source.

POST /datasource/{dataSourceId}/transaction/{transactionId}/

Commit pending transaction.

POST /datasource/{dataSourceId}/translation/

Machine translate portion of game data. First language in the list is a source language.

POST /datasource/{dataSourceId}/validity/

Validadate data source with specified requirements/parameters.

PUT /datasource/{dataSourceId}/

Restore data source from specified documents.

PUT /datasource/{dataSourceId}/collection/{schemaNameOrId}/

Create document.

PUT /datasource/{dataSourceId}/collection/{schemaNameOrId}/documents/

Bulk change documents.

PUT /datasource/{dataSourceId}/collections/

Import documents into multiple collections.

PUT /datasource/{dataSourceId}/transaction/

Wait for data source availability and begin new transaction. Identifier specified in request later could be used with other request in *transaction* parameter.

DELETE /datasource/{dataSourceId}/collection/{schemaNameOrId}/

Delete document by it's id.

DELETE /datasource/{dataSourceId}/completion/thread/{threadId}/

Delete existing AI chat thread.

DELETE /datasource/{dataSourceId}/transaction/{transactionId}/

Reject pending transaction.

/membership

GET /membership/packages/

Get all membership packages.

/notification

GET /notification/

Subscribe on notifications from server. This is WebSocket endpoint, any non 'Upgrade' requests will fail.

/preferences

GET /preferences/

Get default preferences.

PUT /preferences/

Save default preferences.

DELETE /preferences/user/

Reset all user's preferences.

PATCH /preferences/

Patch default preferences.

/project

GET /project/

Get all available projects.

GET /project/my/

Get current user's projects.

GET /project/{projectId}/

Get project by id.

GET /project/{projectId}/preferences/

Get project team-shared preferences.

GET /project/{projectId}/preferences/user/

Get project user's preferences.

POST /project/{projectId}/

Update project with new parameters.

POST /project/{projectId}/branch/{branchName}/

Update branch in project.

POST /project/{projectId}/permissions/

Update project permissions.

POST /project/{projectId}/workspace/

Transfer project form one workspace to another.

PUT /project/

Create new project.

PUT /project/{projectId}/branch/

Create branch in project.

PUT /project/{projectId}/branch/{branchName}/

Push branch content into another branch in this project.

PUT /project/{projectId}/members/

Invite another user into project.

PUT /project/{projectId}/preferences/

Save project team-shared preferences.

PUT /project/{projectId}/preferences/user/

Save project user's preferences.

DELETE /project/{projectId}/

Delete project and all related data.

DELETE /project/{projectId}/branch/{branchName}/

Delete branch in project.

DELETE /project/{projectId}/members/

Expel another user from project.

PATCH /project/{projectId}/preferences/

Patch project team-shared preferences.

PATCH /project/{projectId}/preferences/user/

Patch project user's preferences.

/resourceStorage

GET /resourceStorage/{resourceId}/

Get resource metadata by id.

GET /resourceStorage/{resourceId}/data/

Get resource binary data by id.

PUT /resourceStorage/

Create resource.

DELETE /resourceStorage/{resourceId}/

Delete resource by id.

/search

POST /search/

Search for users, projects, workspaces by specified keyword.

/token

POST /token

null

/user

GET /user/

Get all available users.

GET /user/me/

Get current user.

GET /user/{userId}/

Get user by id.

GET /user/{userId}/public/

Get user public profile by id.

POST /user/password-reset/

Change user password by using code from email.

POST /user/public/

Get public profiles of users by their ids.

POST /user/{userId}/

Update user with new parameters.

POST /user/{userId}/invitations/{invitationId}/

Accept invitation.

POST /user/{userId}/login/api-key/

Add API key login to user.

POST /user/{userId}/login/password/

Change user password by using temporary code or old password.

POST /user/{userId}/mfa/email-code/

Configure email-code multi-factor authentication.

PUT /user/

Create user with specified parameters.

PUT /user/password-reset/

Request password reset.

DELETE /user/{userId}/

Strip personal information from user, quit all groups and block any access to this user.

DELETE /user/{userId}/invitations/{invitationId}/

Decline invitation.

DELETE /user/{userId}/login/api-key/

Delete API key login from user.

DELETE /user/{userId}/login/tokens/

Revoke all issues tokens for specified user.

/workspace

GET /workspace/

Get all available workspaces.

GET /workspace/my/

Get current user's workspaces.

GET /workspace/{workspaceId}/

Get workspace by id.

GET /workspace/{workspaceId}/members/

Get workspace members.

GET /workspace/{workspaceId}/preferences/

Get workspace team-shared preferences.

GET /workspace/{workspaceId}/preferences/user/

Get user's workspace preferences.

POST /workspace/{workspaceId}/

Update workspace with new parameters.

POST /workspace/{workspaceId}/quota-usage/

Get workspace quota usage.

PUT /workspace/{workspaceId}/administrators/

Promote member to workspace administrators.

PUT /workspace/{workspaceId}/preferences/

Save workspace team-shared preferences.

PUT /workspace/{workspaceId}/preferences/user/

> Save user's workspace preferences.

DELETE /workspace/{workspaceId}/administrators/

> Demote member from workspace administrators.

PATCH /workspace/{workspaceId}/preferences/

> Patch workspace team-shared preferences.

PATCH /workspace/{workspaceId}/preferences/user/

> Patch user's workspace preferences.