XVNML / XVNML SCLA

User Guide and Technical Documentation

Version 1.00

Revision History

Date	Version	Description	Author
2023-07-13	1.00	Official Documentation	MJ

Table of Contents

1. Introduction

1.1 Purpose of this document

This document is meant to formally introduce XVNML (X-Tensible Visual Novel Mark-Up Language) as a new format for creating stories. The language's structure is similar to that of an HTML or XML document. This document also elaborates on the XVNML Standard Class Library, which is used to make use of the data generated from an XVNML file. Hopefully, this documentation will guide you and inform you of XVNML essentials and best practices. It goes into great detail, such as the many tags that exist within Standard XVNML, as well as their different properties and syntactical structure.

This document also introduces documentation for XVNML2U (a Unity Package with the XVNML Standard Class Library already integrated). So, for those coming from Unity, you'll be able to find information regarding subject matters pertaining to XVNML2U.

Compiling such information into one document will provide the end-user a solid foundation of how to work with XVNML. However, this document does not go in-depth for XVNML2U nor the XVNML SCLA. This document will be updated accordingly with the new features that are released on both NuGet and GitHub. Other sources for documentation will also be provided in this document as well.

1.2 Intended Audience

This documentation is primarily focused on Game Developer and Game Designers looking for a way to integrate a Dialogue System in a way that's most familiar to them. Serializing dialogue data through JSON or XML is a widespread practice. And for Unity Developers, they may even store their data inside a Scriptable object. By creating a Mark-Up Language catered to the story-writing process, this will grant the target demographic, this makes creating your story similar to writing a screenplay.

1.3 Definitions and Acronyms

1.3.1 Definitions

Keyword	Definitions
Allowed Flags	Annotated with "#". Boolean Properties that are allowed on a Tag
Allowed Parameters	Properties that are allowed on a Tag.
Anonymous Cast Signature	The speaker of a Skriptr Line that has not yet be identified.
Cast	A character that speaks a line of dialogue.
Cast Macros	Macros specific to manipulating a Cast.
Cast Signatures	The form in which a Cast takes prior to saying a line of dialogue.
CDPs	Processes that handle their individual sets of dialogue.
Compatible Parent Tags	Children tags that can work when part of a tag of a higher level.
Control Macros	Macros specific to controlling properties of the DialogueWriter
Custom Macros	Macros created by the end-user with their corresponding endpoints.
Debug Macros	Macros made for debugging.
Declarative Declaration	A Skriptr Line starting with a "@"
Declarative Role	The speaker of a Skriptr Line will speak as normal.
DialogueWriter	A static class that handles the display of content, defined in the XVNML SCLA
Interrogative Declaration	A Skriptr Line starting with a "?"
Interrogative Role	The speaker of a Skriptr Line will prompt you and expect a response.
Macro Endpoints	The method or callback that the macro is attached to.
Macros	Single-Unit commands that can take on one or multiple arguments.
Named Lines	A Skripter Line that's been named, denoted by square brackets.
Narrative Cast Signature	A Cast Signature denoting that the Narrator speaks.
Normal Cast Signature	A Cast Signature denoting that a defined Cast Member speaks.
Persistent Cast Signature	A Cast Signature denoting the carrying-over of the previous Cast Member that spoke.
Portrait/Expression	A Cast's Image or Pose in order to convey emotion.
Print (Character) Macros	Special "Insert" macros that print a special kind of character.
Prompt Response	A response from answering a prompt initiated from a Interrogative Declaration.

Prompts	Questions that the speaker proposes and expects a response.
Proxy (Main) File	The first file to be processed.
Reference	Prefixed by "$", references a tag by name, and returns data.
Scene	An image that plays as a background.
Skriptr	A special scripting language used to create dialogue.
Skriptr Role	The role in which a Skriptr Line adheres to (either a Declarative Role, or an Interrogative Role).
Source File	A file that is being source by a Proxy file or another Source file.
Tag	A building-block of an XVNML file (also known as an element).
Tag Flag	A Boolean that changes the behavior or action of a tag.
Tag Occurrence	How often can a tag occur in a file or in-scope of another tag.
Tag Parameter	An argument that can be set on a tag or element.
Tag Value	Any input (excluding children elements) that are in-scope of a tag.
User-Defined Tags	Tags that are created by the user via the XVNML SCLA.
Variable Control Macros	Macros that handle variable declaration, initialization, setting, and getting.
Voice	The audio representation of a defined Cast member.
XVNM2U	A Unity Package with the XVNML SCLA integrated.
XVNML	A mark-up language made for storytellers.
XVNML SCLA	The C# Class Library for XVNML using .Net Standard 2.1 (to assure compatibilities with applications (e.g., Unity Engine).

1.3.2 Acronyms, Symbols, and Abbreviations

Acronym, Symbol, or Abbreviations	**Definitions**
$	Reference
?	Interrogative Declaration
@	Declarative Declaration
<	New Line
<<	End of Dialogue / Prompt Response
SCLA	Standard Class Library API

XVNML	X-Tensible Visual Novel Mark-Up Language
XVNML2U	XVNML To Unity
CDPs	Concurrent Dialogue Processes

1.4 Naming Conventions

1.4.1 Files

When referring to a Main or Proxy file, name your file, followed by ".main.xvnml" or ".proxy.xvnml".

When referring to a Source file, name your followed, followed by the type that the source is based on. E.g. ".dialogue.xvnml", ".cast.xvnml", ".image.def.xvnml", ".scene.xvnml", ".audio.def.xvnml", etc.

1.4.2 Folders

When using files as references, be sure you have a directory with the source's type in mind. This is because when setting the "src" tag parameter of a tag that has it allowed, by default it'll search at its corresponding directory to find it. Here's a table mapping out the source file extension to the directory that it should be under.

Keyword	Directory
Audio	[Main File Root Directory]/Audio
Cast	[Main File Root Directory]/Casts
Dialogue	[Main File Root Directory]/Dialogue
Image	[Main File Root Directory]/Images
Macro	[Main File Root Directory]/Macros
Scene	[Main File Root Directory]/Scenes

1.4.3 Macros

Pre-established macros and macros that are created by the end-user has the “snake-case” naming convention. Macros are denoted by surrounded curly brackets.

```
@ {move_cast::("Hana", -750)}We can move her to the left...<
@ {move_cast::("Hana", 750)}Move her to the right...<
@ {move_cast::("Hana", 0)}Put her back in center...<
```

1.4.4 Cached Macros

The name of macros that are cached under the “Macro Cache” tag are prefixed with an underscore. The usage of these cached macros are referenced (which reference identifiers are always prefixed by a dollar sign).

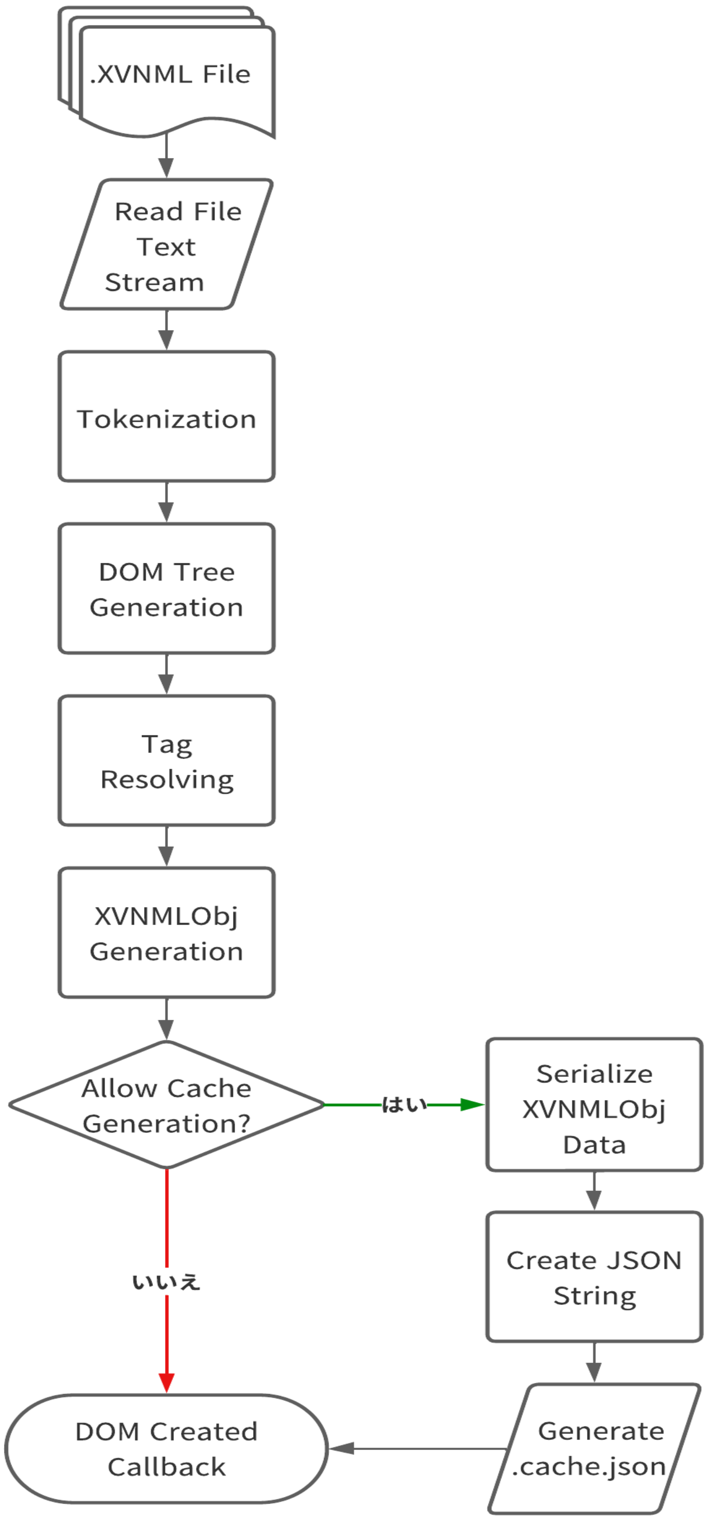
```
["Main-FirstTime"]
? So, what would you like to know about XVNML or about XVNLM2U?{$_pauseAndHide}>>
(
        ("What is XVNML?")>                 @ {$_chapter1}<<
        ("Why XVNML?")>                     @ {$_chapter2}<<
        ("What is XVNML2U...")>             @ {$_chapter3}<<
        ("Integrate XVNML Manually...")>    @ {$_chapter4}<<
        ("Some Best Practices...")>         @ {$_chapter5}<<
)

["Main"]
@ And with that said...<
? Is there anything else you would like to know?{$_pauseAndHide}>>
(
        ("What is XVNML?")>                 @ {$_chapter1}<<
        ("Why XVNML?")>                     @ {$_chapter2}<<
        ("What is XVNML2U...")>             @ {$_chapter3}<<
        ("Integrate XVNML Manually...")>    @ {$_chapter4}<<
        ("Some Best Practices...")>         @ {$_chapter5}<<
        ("End Demo")>                       @ {$_conclude}<<
)
```

2. General Overview

2.1 Technologies Used

- C#
- .NET Standard 2.1
- NewtonSoft.Json
- Visual Studio Code (for Syntax Highlighting Extension)
- Unity

2.2 General Process



3. Technical Requirements

- XVNML.dll
- Visual Studio 2022
- Unity Game Engine
- Visual Studio Code
- XVNML2U*
- XVNMLExt VSCode Extension

3.1 XVNML.dll (XVNML SCLA)

The XVNML SCLA is avaliable on NuGet, as well as on our GitHub page for XVNML.

3.2 Visual Studio 2022

You can look for instructions on how to download Visual Studio 2022 onto your machine here.

3.3 Unity

You can download the Unity Game Engine here. This will also give you the option to download Visual Studio 2022, which may save you some time. It may also do it for Visual Studio Code as well.

* As of the time of writing this, XVNML2U is still in development. This will change when the tool is made public.

3.4 Visual Studio Code

You can download Visual Studio Code here. Be sure to also install the extension for XVNML (XVNMLExt) in which provided you syntax highlighting, as well as some snippets here.

4. XVNML DOM Structure

4.1 Root Tags

Every .xvnml file has a root tag. This root tag will hold all information regarding their children, and the data that they contain. The root tag will all let the XVNML SCLA know either it's the file is used as the "Main" file, or that a file is being used a source being referenced from the "Main" file.

4.1.1 Proxy Root

A Proxy Root is defined by the top-most tag <proxy>. This will hold specific target data, as well as other properties such as the canvas width and height for any style-based mark-ups. This is the first file to be read (and should be the first file read or accessed) from the XVNML SCLA.

```
<proxy name::"ConsoleApp.Main.XVNML" screenWidth::1920 screenHeight::1080>
        <metadata name::"_metadata">
                <title name::"XVNMLEssentials"/>
                <date value::"06/25/2023"/>
                <copyright year::2023 owner::"Enjyoii"/>
                <tags #useChildren>
                        <tag name::"XVNML"/>
                        <tag name::"C#"/>
                        <tag name::".NET"/>
                        <tag name::"Unity"/>
                        <tag name::"Educational"/>
                </tags>
        </metadata>

        <castDefinitions>
                <cast name::"Raven"     src::"Raven/Raven.cast.xvnml"       />
                <cast name::"Maple"     src::"Maple/Maple.cast.xvnml"       />
                <cast name::"Hanabi"    src::"Hanabi/Hanabi.cast.xvnml"     />
                <cast name::"Devi"      src::"Devi/Devi.cast.xvnml"         />
                <cast name::"Silka"     src::"Silka/Silka.cast.xvnml"       />
                <cast name::"Hana"      src::"Hana/Hana.cast.xvnml"         />
                <cast name::"Ageha"     src::"Ageha/Ageha.cast.xvnml"       />
                <cast name::"Konoha"    src::"Konoha/Konoha.cast.xvnml"     />
                <cast name::"Urara"     src::"Urara/Urara.cast.xvnml"       />
                <cast name::"Anemelle"  src::"Anemelle/Anemelle.cast.xvnml" />
                <cast name::"Saki"      src::"Saki/Saki.cast.xvnml"         />
                <cast name::"Luu"       src::"Luu/Luu.cast.xvnml"           />
                <cast name::"Mythril"   src::"Mythril/Mythril.cast.xvnml"   />
                <cast name::"Adula"     src::"Adula/Adula.cast.xvnml"       />
        </castDefinitions>

        <sceneDefinitions>
                <scene name::"OutsideOfSchool"
src::"School\OutsideOfSchool.scene.xvnml"/>
        </sceneDefinitions>

        <keycodeDefinitions name::"_globalInputDef">
                <keycode
name::"_proceedWithMouse"     vkey::LeftMouse      purpose::PROCEED/>
                <keycode
name::"_proceedWithEnter"     vkey::Return         purpose::PROCEED/>
                <keycode
name::"_navUp"                vkey::Up             purpose::NAVIGATION_UP/>
                <keycode
name::"_navDown"              vkey::Down           purpose::NAVIGATION_DOWN/>
```

```
                <keycode
name::"_navLeft"               vkey::Left           purpose::NAVIGATION_LEFT/>
                <keycode
name::"_navRight"              vkey::Right          purpose::NAVIGATION_RIGHT/>
        </keycodeDefinitions>

        <macroCache>
                <macro name::"_runTest" symbol::"var">
                    <arg value::"love"/>
                    <arg value::200/>
                </macro>
                <macro name::"_think">
                   <macro arg::500 symbol::"delay"/>
                   <macro arg::"." symbol::"insert"/>
                   <macro arg::500 symbol::"delay"/>
                   <macro arg::"." symbol::"insert"/>
                   <macro arg::500 symbol::"delay"/>
                   <macro arg::"." symbol::"insert"/>
                </macro>
                <macro name::"_awkwardPause">
                    <macro arg::500 symbol::"delay"/>
                    <macro arg::"." symbol::"insert"/>
                    <macro arg::500 symbol::"delay"/>
                    <macro arg::"." symbol::"insert"/>
                    <macro arg::500 symbol::"delay"/>
                    <macro arg::"." symbol::"insert"/>
                    <macro arg::500 symbol::"delay"/>
                    <macro arg::"." symbol::"insert"/>
                    <macro arg::500 symbol::"delay"/>
                    <macro arg::"." symbol::"insert"/>
                    <macro arg::500 symbol::"delay"/>
                    <macro arg::"." symbol::"insert"/>
                    <macro arg::1000 symbol::"delay"/>
                </macro>
                <macro name::"_defaultTextSpeed">
                    <macro arg::40 symbol::"set_text_speed"/>
                    <macro symbol::"pass"/>
                </macro>
        </macroCache>
        <dialogue name::"MainTest" src::"Test/test0.dialogue.xvnml"/>
        <dialogue name::"Osaka's Dream" src::"Test/test1.dialogue.xvnml"/>
</proxy>
```

4.1.2 Source Root

A Source Root is defined by the top-most tag <source>. This file will be read if a proxy file as for it. As the Proxy file is being read, it'll resolve and find the Source file that it's looking for, undergoing the usual Tokenization and Parsing and Tag Resolving processes.

```
<source name::"Test1.Dialogue.XVNML">
    <dialogue name::"Osaka's Dream">
        @ {$_defaultTextSpeed}<

        ? Do you wish to know about Osaka's Dream?>>
        (
            $> If yes, play Osaka's Dream Scene from Azumanga Daioh
            ("Yes")> @ {pass}<<

            $> Otherwise, end the dialogue sequence.
            ("No")> @ Your lose, my dude...{end}<<
        )

        @ChiyosDad Herro Everynyan!<
        @* How are you... Fine ssank you...<

        @Osaka OH MAI GAAAAH!!!!<

        @ChiyosDad I wish I were a bird.<

        @Osaka なんであんた英語喋っとる？<

        @ChiyosDad 娘がアメリカに行くのです。<

        @Osaka 鳥合ったら？<

        @ChiyosDad 鳥でわない。わはいは猫である。<
        @* 猫なければ、なんだと意味のだ？<

        @Osaka あんたの顔なんか見に取るな。。。<

        @ChiyosDad {$_awkwardPause}<

        @Osaka 森、先首相。。。<

        @ChiyosDad {aster|insert::"nuhuh"|aster}<

        @Osaka I'mmu sorreeeee!<
        @ {aster|insert::"Famous cat appears"|aster}<

        @ChiyosDad Hmmmmm...<
        @* お前が本当猫かね？<
        @* 私は猫舌がどうだ。猫いよー！<

        @Okasa ああ！あたしも猫舌。<

        @ChiyosDad Then I shall pull out your tongue.<

        @Osaka ええええええええ?! なんでーーーーー？！<
    </dialogue>
</source>
```

4.2 Tag Properties

Tag properties are data that a particular tag type has available. There are two types of tag properties: Parameters, and Flags.

4.2.1 Tag Parameters

Parameters can be given a single argument or reference of another tag via its name. It takes in a number, a string, or reference.

By default, a tag will have a “name” parameter, “altNames” parameter, as well as an “id” parameter. Both are optional unless you need to reference the element or tag in the document itself or utilizing the “XVNML.Utilities” namespace in the SCLA.

```
$> Properties such as "name", "lang", and "engine" are Tag Parameters.
<proxy name::"Test.xvnml" lang::"CSharp" engine::"Native">
    <dialogue name::"Main">
        @Me Hello!<
        @* What is going on?<
        @* I have no idea what I'm doing right now.<
    </dialogue>
</proxy>
```

4.2.2 Tag Flags

A flag is a declaration of a tag type that enables it to do certain functions. I can also be used to alter its default behavior.

```
<metadata name::"_metadata">
      <title name::"XVNMLEssentials"/>
      <date value::"06/25/2023"/>
      <copyright year::2023 owner::"Enjyoii"/>
      <tags #useChildren>
            <tag name::"XVNML"/>
            <tag name::"C#"/>
            <tag name::".NET"/>
            <tag name::"Unity"/>
            <tag name::"Educational"/>
      </tags>
</metadata>
```

4.3.2 Tag Values

Tag values are separate from Tag Parameters. However, sometimes a Tag Parameter may substitute for the lack of a tag value.

4.3.3 Skriptr

A valid tag value that's exclusive to the <dialogue> tag type is inputing Skriptr Lines. A Skriptr Line may start with an "@" (denoting it takes on a Declarative Role), or a "?" (denoting it takes on a Interrogative Role). The difference between these two roles is that with a line in the Interrogative Role, your Skriptr line is then known as a "prompt" in which the user must response to.

4.3.3.1 Skriptr: Cast Signature

The syntax between the Role Symbol and a whitespace is known as the Cast Signature. There are things such as a Full Cast Signature, a Partial Cast Signature, a Full Inversed Cast Signature, and an Partial Inversed Cast Signature. This may also be referred to as "Cast Notation"

4.3.3.2 Skriptr: Narrative Cast Signature

A Narrative Cast Signature is formed in the following notations:

Cast Notation	Form
@ Dialogue	Partial Narrative Cast Signature
@>V::Voice Dialogue	Full Narrative Cast Signature

The Partial Narrative Cast Signature is the most common notation you'll use for most of your dialogue with a Narrator as the speaker. Narrator (in this case) is that no (cast member) name is defined, thus is something or someone speaking in the Third or First Person Perspective.

4.3.3.3 Skriptr: Normal Cast Signature

A Normal Cast Signature is the normal form of notation. The signature can be formed in the following notations:

Cast Notation	Form
@John	Empty Normal Cast Signature
@John>Happy @John>E::Happy	Expressive Normal Cast Signature Partial Normal Cast Signature (Expressive)
@John>V::Voice	Inversed Normal Cast Signature Vocal Normal Cast Signature Partial Normal Cased Signature (Vocal) Inversed Partial Normal Signature
@John>Happy>Voice	Full Normal Normal Cast Signature
@John>V::Voice>E::Happy	Full Inversed Normal Cast Signature

If the <castDefinitions> have a list of defined <cast> elements, you can change the image that represents the Cast’s Expression (also know as the Cast’s Portrait). If they have voices attached to a cast, you’ll have data representing the audio file that voices a cast member.

```
<source name::"Hana.Cast">
    <imageDefinitions>
        <image name::"HanaSmile"        src::"HanaSmile.png"    pathMode::Relative/>
        <image name::"HanaHappy"        src::"HanaHappy.png"    pathMode::Relative/>
        <image name::"HanaSad"          src::"HanaSad.png"      pathMode::Relative/>
        <image name::"HanaAngry"        src::"HanaAngry.png"    pathMode::Relative/>
        <image name::"HanaNeutral"      src::"HanaNeutral.png"  pathMode::Relative/>
    </imageDefinitions>
    <audioDefinitions>
        <audio name::"HanaWow"          src::"HanaWow.wav"      pathMode::Relative/>
    </audioDefinitions>
    <castDefinitions>
        <cast name::"Hana">
            <portraitDefinitions>
                <portrait name::"Smile"     img::$HanaSmile/>
                <portrait name::"Happy"     img::$HanaHappy/>
                <portrait name::"Sad"       img::$HanaSad/>
                <portrait name::"Angry"     img::$HanaAngry/>
                <portrait name::"Neutral"   img::$HanaNeutral/>
            </portraitDefinitions>
            <voiceDefinitions>
                <voice name::"Wow"  audio::$HanaWow/>
            </voiceDefinitions>
        </cast>
    </castDefinitions>
</source>
```

```
["P1-Skriptr-CastSignatures-Normal"]
@ {$_defaultTextSpeed}A Normal Cast Signature denotes that a character is speaking.<
@Jimmy You can put whatever name in here...<
@* However, if that cast was not defined, you can't using expressions
 or voices.<
@Hana {cue_cast::"Center"}However for me, I am a defined Cast member!<
@*>Happy I can change expressions {paren}as you saw earlier...{paren_end}<
@*>V::Wow And I can play a voice based on the how the Normal Cast
 signature is formed.<
@ {exp::("Hana","Smile")}Nice work, Hana. I'm
really{exp::("Hana","Happy")|react::("Hana","Jump")} proud of you.<
@Hana Hehe!<
```

4.3.3.4 Persistent Cast Signature

The Persistant Cast Signature will carry over the cast that's currently speaking, preventing you from re-entering the same name on each new line. This does not work on Narrative Cast Signatures or Anonymous Cast Signatures. The signature can take on one of the following notations:

Cast Notation	Form
@*	Empty Persistent Cast Signature
@*>Happy @*>E::Happy	Expressive Persistent Cast Signature Partial Persistent Cast Signature (Expressive)
@*>V::Voice	Inversed Persistent Cast Signature Vocal Persistent Cast Signature Partial Persistent Cased Signature (Vocal) Inversed Persistent Normal Signature
@*>Happy>Voice	Full Persistent Persistent Cast Signature
@*>V::Voice>E::Happy	Full Inversed Persistent Cast Signature

4.3.3.5 Anonymous Cast Signature

The Anonymous Cast Signature is a special cast in which a cast member has not yet been introduced, or that their name is unknown. This is different from the Narrative Cast in that the "Unknown Cast" is a valid Cast Member that's been predefined by the XVNML SCLA. If you were to use the "speaker" macro, you'll print a "???". The signature can take on one of the following notations:

Cast Notation	Form
@???	Empty Normal Cast Signature
@???>Happy @???>E::Happy	Expressive Normal Cast Signature Partial Normal Cast Signature (Expressive)
@???>V::Voice	Inversed Normal Cast Signature Vocal Normal Cast Signature Partial Normal Cased Signature (Vocal) Inversed Partial Normal Signature
@???>Happy>Voice	Full Normal Cast Signature
@???>V::Voice>E::Happy	Full Inversed Cast Signature

4.3.3.6 Interrogative Skriptr Lines

The charts prevently seen where Cast Notations in their Declarative Roles. These notations can change into an Interrogative Role by substituting the “@” for a “?”. However, the syntax for Skriptr Lines taking the Interrogative Role requires some extra steps demonstrated here:

```
?*>Smile What do you want to do today?>>
(
        ("Create more visual novels...")>
        @Hana>Happy Good choice!<
        @* I actually have a really great story idea!<
        @* Want to hear it?<<

        ("Hang out with you...")>
        @Hana>Happy>Wow {react::"Jump"}Hehe!<
        @* That's very sweet!<
        @* Let's hang out then!<<

        ("Sleep...")>
        @Hana You must be tired then.<
        @* I'll always be here if you want to do anything!<<

        ("Can I hear your Japanese?")>
        @Hana>Happy>Wow {sts::20|react::"Jump"}いいよ！<
        @* どうですか？<
        @*>Smile 日本語で話してること。。。{del::1000}へんなの？<
        ?* 何か話し事がありますか？>>
        (
                ("Tell me how you learned Japanese?")>
                @Hana まあ、実はね。。。<
                @* 私が日本人ので、日本語で話せるよ！<
                @* 母は日本人と父さんはイギリス人です。<
                @* でもね、米国には１０年くらいで暮らしていて、全部人生で英語ではとんど話してばかりで
す。<
                @* おかしいようね？<<

                ("What's your favorite food?")>
                @Hana 沢山食べ物が好きよ！<
                @* けど、好きな食べ物は。。。{del::1000|exp::"Happy"|clr}大福です！<
                @*>Smile とても上手くて、美味しくて。。。<
                @*>Happy それが全部を一気に食べられるも！<
                @*>Smile すべての味は好き！<
        )

        ?Hana {sts::45}Did you happen to understand all that?>>
        (
                ("Yes!")>
                @Hana>Happy>Wow {react::"Jump"}I'm so glad.<<

                ("A little bit...")>
                @Hana Well, my Japanese is a bit off...<
                @* I only speak Japanese around my mom...<
                @* But I've spend most of my life with my British father in the United 
States.<
                @* So my Japanese is very rusty!<
                @*>Happy I do appreciate you listening to me though!<<

                ("Not exactly...")>
                @Hana>Happy>Wow Hehe!<
                @* That's okay!<
                @*>Smile My Japanese has been rusty over my time in the United States.<
                @* So you have nothing to worry about.<<
        )
)
```

5. XVNML Standard Tags

XVNML has a list of Standard Tags that will be common throughout every facet of your development workflow. The XVNML SCLA already parses and resolves these tags and generates various kinds of data to be used.

As of the typing of this documentation, there are currently 32 pre-defined tags that you can use in XVNML, each with their own sets of properties.

Keyword	Compatibility	Occurrence	Parameters	Flags
Arg	Macro	Multiple	“value”	NA
Audio	AudioDefinitions	Multiple	“pathMode” “src”	NA
AudioDefinitions	Proxy Source	PragmaOnce	NA	NA
Author	Metadata	PragmaOnce	NA	NA
Cast	CastDefinition Source	Multiple	“src”	NA
CastDefinitions	Proxy, Source	PragmaOnce	NA	NA
Copyright	Metadata	PragmaOnce	“owner” ”year”	NA
Date	Metadata	PragmaOnce	“value”	NA
Dependency*	DependencyDefinitions	Multiple	NA	NA
DependencyDefinitions*	Proxy, Source	PragmaOnce	NA	NA
Description	Metadata	PragmaOnce	“content”	NA
Dialogue	Proxy Source DialogueGroup	Multiple	“pathMode” “src”	“dontDetain” ”textSpeedControlledExternally
DialogueGroup	Proxy Source	Multiple	NA	“actAsSceneController”
Image	ImageDefinitions Source	Multiple	“pathMode” “src”	NA
ImageDefinitions	Proxy Source	PragmaOnce	NA	NA
Keycode	KeycodeDefinitions Source	Multiple	“vKey” “purpose”	NA
KeycodeDefinitions	Proxy Source	PragmaOnce	NA	NA
Macro	MacroCache	Multiple	“arg”	NA

	Macro		"symbol"	
MacroCache	Proxy Source	PragmaOnce	NA	NA
Metadata	Proxy	PragmaOnce	NA	NA
Portrait	PortraitDefinitions Source	Multiple	"image"	NA
PortraitDefinitions	Cast	PragmaLocalOnce	NA	NA
Proxy	NA	PragmaOnce	"engine" "target" "lang" "screenWidth" "screenHeight"	NA
Scene	SceneDefinitions	Multiple	"img" "src"	NA
SceneDefinitions	Proxy Source	PragmaOnce	NA	NA
Source	NA	PragmaOnce	NA	NA
Tag	Tags	Multiple	NA	NA
Tags	Metadata	PragmaOnce	"list"	"useChildren"
Title	Metadata	PragmaLocalOnce	NA	NA
Url	Metadata	PragmaOnce	NA	NA
Voice	VoiceDefinitions	Multiple	"audio"	NA
VoiceDefinitions	Cast	PragmaLocalOnce	NA	NA

6. XVNML Standard Macros

Macros are single-unit commands that can be called while a Cast is speaking. They can take in an argument, multiple arguments, or no arguments at all. XVNML provides you with numerous amounts of Standard Macros that you can use during your development process.

```
["P1-Skriptr-CorrectAnswer"]
@ {$_defaultTextSpeed}There are 50 states that makes up the United States of America.<
@ Awesome huh?<
@ Let's take a look at the file where we have all of this dialogue.<
@ And don't freak out when you look at it.<
@ I promise you, you'll understand it faster than I can finish this sentence.<
@ Search up and find a file called {quot}XVNMLEssentials.main.xvnml{quot}.<
@ Open that in VSCode and go to Line 61.<
@ You will see that that's the first thing that we started off with.<
@ And if you jump to Line 194, you'll exactly what I'm currently saying.<
@ You notice that it's just the {quot|at|quot} symbol when stating a
Declarative Role on a Skriptr line.<
@ However, if you've defined Cast members within your XVNML project...<
@ You can set them as a speaker!<
@ {$_initializeCastMotionSettings}For example...<
@??? {cue_cast::("Hana","Center")}Hello!<
@??? My name is Hana!<
@Hana>Happy {react::"Jump"}It's very nice to meet you!<
@ Anything after and before actual dialogue is known as the Cast Signature.<
@ {exp::("Hana", "Smile")|react::("Hana", "Jump")}If you look at Line 204, this is
considered a Normal Cast Signature.<
@ It's the most common signature that you'll see inside an XVNML document.<
@ Anything without a name is considered a Narrative Cast Signature.<
@ As the name suggests, it's meant to denote that the Narrator is speaking.<
@??? If there's a {quot|qm|qm|qm|quot} in place of the name...<
@??? This is known as the Anonymous Cast Signature.<
@* And anything with an {quot|aster|quot} is consider a Persistent Cast Signature.<
@ The general format of a Cast Signature is the following...<
@
{paren}{at}{pipe}{qm}{paren_end}{tag_end}{brack}Expressions{brack_end}{tag_end}{brack}
Voice{brack_end}<
@ {move_cast::("Hana",-1325)}It's definitely a lot to keep up with, but that's what
the XVNMLEssentials guide is for...<
@ To help you get in touch with XVNML.<
```

Control Macros			
Keywords	**Symbols**	**Signatures**	**Description**
Delay	“delay”	uint	Pause the DialogueWriter for {nth} amount of milliseconds.
Insert	“insert” “ins”	string	Print a character or symbol.
Set Text Speed	“set_text_speed” “sts”	uint	Set the text speed for the DialogueWriter.
Clear	“clear” “clr”	void	Clear displaying text.
Pause	“pause”	void	Wait for user feedback.
Pass	“pass”	void	Do not wait for user input upon line completion.
Operation*	“op”	string	Perform an operation based on expression.
Jump To	“jump_to”	uint string	Jump to a line by name or by index.
Lead To	“lead_to”	int	Relatively go to the next line, or the previous line by {nth} amount.
End	“end”	void	End dialogue sequence.

Debug Macros			
Keywords	**Symbols**	**Signatures**	**Description**
Show Process ID	“process_id” “pid”	void bool	Display what process the dialogue is running on.
Show Cursor Index	“curdex”	void bool	Display the current position of the cursor when text is displayed.
Show Line Index	“lindex”	void bool	Show what index a line is.

Print Macros			
Keywords	**Symbols**	**Signatures**	**Description**
New Line	“new_line” “nl” “n”	void	Print a new line.
Tab	“tab” “tb” “t”	void	Print a Tab.
Space	“space” “ws” “w”	void	Print a space.
Hash	“hash”	void	Print a #
Parenthesis Open	“paren” “p”	void	Print a (
Parenthesis Close	“paren_end” “/p”	void	Print a)
Quotation	“quot”	void	Print a “
Curly Bracket Open	“curly”	void	Print a {
Curly Bracket Close	“curly_end” “/curly”	void	Print a }
Square Bracket Open	“brack”	void	Print a [
Square Bracket Close	“brack_end” “/brack”	void	Print a]
Pipe	“pipe”	void	Print a |
Asterisk	“asterisk” “aster”	void	Print a *
Ampersand	“ampersand” “amper”	void	Print a &
Hat	“hat”	void	Print a ^
Tilda	“tilda”	void	Print a ~
Forward Slash	“forward_slash” “slash”	void	Print a /

Backslash	“backward_slash” “blash”	void	Print a \
Semicolon	“semicolon” “semi”	void	Print a ;
Colon	“colon”	void	Print a :
Tag Open	“tag”	void	Print a <
Tag Close	“tag_end” “/tag”	void	Print a >
Percent	“percent” “per”	void	Print a %
Plus	“plus”	void	Print a +
Equals	“equals”	void	Print a =
At	“at”	void	Print a @
Question Mark	“question_mark” “qm”	void	Print a ?
Trademark	“trademark” “trade”	void	Print a ™
Copyright	“copyright” “copy”	void	Print a ©
Register	“registermark” “reg”	void	Print a ®
Bullet Point 1	“bullet_style_1” “bul1”	void	Print a •
Bullet Point 2	“bullet_style_2” “bul2”	void	Print a ◦
Bullet Point 3	“bullet_style_3” “bul3”	void	Print a ‣
Ellipsis	“ellipsis” “ell”	void	Print a …
Section	“section” “sec”	void	Print a §
Degree	“degree”	void	Print a °

	“deg”		
Plus-Minus	“plus_minus” “pm”	void	Print a ±
Speaker	“speaker” “sp”	void	Print out the name of the current speaker.

Cast Macros			
Keywords	**Symbols**	**Signatures**	**Description**
Set Expression	“expression” “portrait” “exp” “port”	int string	Set a Cast Member’s expression.
Set Voice	“voice” “vo”	int string	Set a Cast’Member’s voice upon speaking.

Variable Control Macros (Not Implemented)			
Keywords	**Symbols**	**Signatures**	**Description**
Declare Variable	“declare”	{string, object}	Declare and initialize a variable.
Set Variable	“set”	{string, object}	Set a variable.
Get Variable	“get”	string	Get a variable.

7. (For Unity Developers) XVNML2U Specific Tags (TBD)

8. (For Unity Developers) XVNML2U Specific Macros

Utilizing the feature of creating Custom Macros, XVNML2U provides you with Unity-Specific macros that can be used in combination with the Standard XVNML macros.

Control Macros			
Keywords	**Symbols**	**Signatures**	**Description**
Debug	“debug”	string	
Hide Text Box	“hide_text_box” “htb”	void	Make the text box invisible.
Show Text Box	“show_text_box” “stb”	void	Make the text box appear.

Sound Macros			
Keywords	**Symbols**	**Signatures**	**Description**
Play Sound	“play_sound” “plsnd”	void	Play a sound that’s been set.
Pause Sound	“pause_sound” “pssnd”	void	Pause a sound that’s been set.
Set Sound	“set_sound” “ssnd”	string	Set a sound to play.
Set Sound Volume	“set_sound_volume” “ssndv”	uint	Set sound volume prior to playing.
Loop Sound	“loop_sound” “loop”	bool	Toggle on sound loop.
Enable Sound Loop	“enable_sound_loop” “esndl”	void	Enable sound looping.
Disable Sound Loop	“disable_sound_loop” “dsndl”	void	Disable sound looping.
Play Sound Effect	“play_sound_effect” “play_sfx” “psfx”	{string, uint}	Play a sound effect.

Cast Macros			
Keywords	**Symbols**	**Signatures**	**Description**

Cue Cast	“cue_cast”
	“cast”
		string
		{string, uint}
		{string, string}
		{string, string, uint}
			Have a Cast Member appear on the Left, Center, or Right part of the stage with an offset.

Move Cast	“move_cast” “mcst”	int {string, int}	Move the target Cast Member.
Set Cast Motion	“set_cast_motion” “scstm”	string	Set how Cast Members Move.
Set Cast Motion Duration	“set_cast_motion_duration” “scstmd”	float	Set how long the motion is for Cast Members.
Cast Enters From	“cast_enters_from” “cstef”	string	Set where the target Cast Member enters from.
Cast React	“react”	string {string, string}	Have the Cast Member react.

Scene Macros			
Keywords	**Symbols**	**Signatures**	**Description**
Use Scene	“use_scene” “scene”	string {string, int}	Load a scene on the background at a specified layer.
Clear All Scenes	“clear_all_scenes” “clrascn”	void	Clear all scenes from the background.
Clear Scene	“clear_scene” “clrscn”	void string uint {string, uint}	Clear a loaded scene, a scene by name, or by layer.

Standard Macro Overrides

Keywords	Symbols	Signatures	Description
Set Expression	"expression" "portrait" "exp" "port"	{string, string} {string, int}	Set the Cast Member's expressions.
Set Voice	"voice" "vo"	{string, string} {string, int}	Set the Cast Member's voice upon speaking.

9. XVNML SCLA: Constructing User-Defined Tags

You can use the XVNML SCLA to create your own Custom or User-Defined Tags. You can have your tag to about anything: configurations, data, even be referenced for macros if it allow it. Be sure to read the ***XVNML SCLA Scripting*** documentation for a better understand of each class and their properties

DISCLAIMER As of the typing of this document, the XVNML SCLA is only compatible with C#. So all code examples will be in C#. In the future, There will be another version of XVNML called the XVNML Native API, which will allow other languages to utilize these same concepts.

9.1 Tag Association

Behind the scenes, each tag or element in a XVNML document has an assoication with a tag deriving from the TagBase abstract class. The attribute "AssociateWithTag" is responsible for mapping out tag elements inside your XVNML file to it's C# counterpart. This can be accessed using the "XVNML.Core.Tags" namespace.

9.1.1 Tag Name

The first field inside the AssociateWithTagAttribute will be the tag name. This will let XVNML know that the class below is associated with a tag of this name. When during the tokenization and parsing process, it will check the XVNML file and all of its element to make sure there are registered and valid associations.

9.1.2 Setting Parent Tag Compatibility

You can set one or multiple types that your tag is compatible with. Again, XVNML will varify that your tag is a child of those particular tags. If there is no parent tag, it is treated as Root Tag. If the element's parent inside the XVNML file doesn't match up to the attribute set for you User-Defined tag, you'll get an error.

9.1.3 Setting Tag Occurrence

You can set how often a tag should appear in a document. This helps give a level of consistency and control as your are building your XVNML document.

Keywords	TagOccurance	Description
PragmaOnce	TagOccurance.Pragma	A Tag should only occur once in a document.
PragmaLocalOnce	TagOccurance.PragmaLocalOnce	A Tag should only occur once within the scope of another tag.
Multiple	TagOccurance.Multiple	A Tag can appear multiple times inside a document.

9.2 "OnResolve" Method

The "OnResolve" method is a protected method for all classes derived from the "TagBase" abstract class. It is called after the analyzing and parsing of a tag has been completed. This is where you implement the logic of the tag upon receiving information from the parsing process. You can also configure the parameters and flags that your custom tag is allowed to use prior to resolving the tag completely.

9.3 "GetParamter" Method

If the tag inside your XVNML file satisfies its Allow Parameters and Flags, you can receive more information on your tag property by using "GetParameter".

9.4 "GetParamterValue<T>" Method

This is the same as "GetParameter" but reduces the need to use the dot operater to access the value of that parameter (it's all done for you).

9.5 "HasFlag" Method

Use this method to see if a tag has one of its Allowed Flags attached to it.

9.6 “Collect<T>” Method

Allows you to collect mutiple children elements of type T.

9.7 “SearchElement<T>” Method

Allows you to do a deep search of a child element of type T. Because it goes into every part of it’s children element to find it’s target one, this method can be expensive when invocated. Use it cautiously, and be sure to cache your data after searching.

9.8 “GetElement<T>” Method

Allows you to grab a child element of type T.

9.9 Setting Allowed Parameters

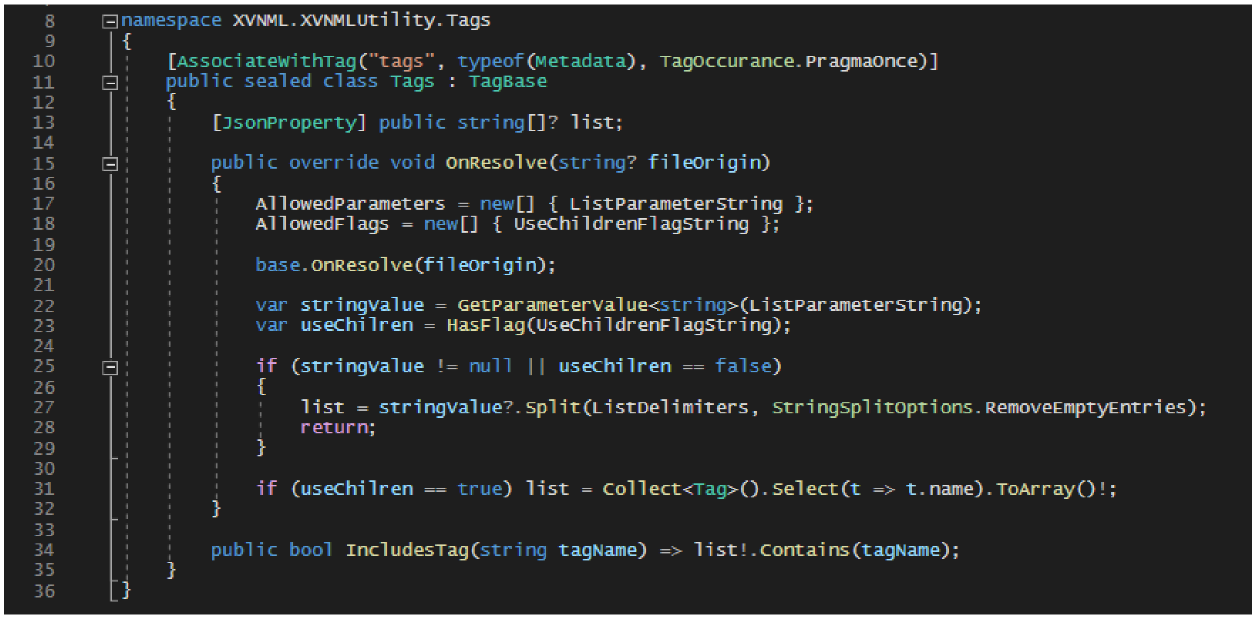
A protected property of TagBase that allows you to specify possible parameters for your tag.

9.10 Setting Allowed Flags

A protected property of TagBase that allows you to specify possible flags for your tag.

9.11 User-Defined Tag Example

This is the logic implement for the <tags> element within the XVNML SCLA, exemplifying some of the previous topics discussed. This tag is only compatible with the <metadata> element, and only occurs once in the whole document. See more at 5. XVNML Standard Tags.



```
namespace XVNML.XVNMLUtility.Tags
{
    [AssociateWithTag("tags", typeof(Metadata), TagOccurance.PragmaOnce)]
    public sealed class Tags : TagBase
    {
        [JsonProperty] public string[]? list;

        public override void OnResolve(string? fileOrigin)
        {
            AllowedParameters = new[] { ListParameterString };
            AllowedFlags = new[] { UseChildrenFlagString };

            base.OnResolve(fileOrigin);

            var stringValue = GetParameterValue<string>(ListParameterString);
            var useChilren = HasFlag(UseChildrenFlagString);

            if (stringValue != null || useChilren == false)
            {
                list = stringValue?.Split(ListDelimiters, StringSplitOptions.RemoveEmptyEntries);
                return;
            }

            if (useChilren == true) list = Collect<Tag>().Select(t => t.name).ToArray()!;
        }

        public bool IncludesTag(string tagName) => list!.Contains(tagName);
    }
}
```

10. XVNML SCLA: Constructing Custom Macros

You can utlizize the XVNML SCLA to construct your own Custom Macros using C#. This will require to using the “XVNML.Utilities.Macros” namespace. Using the MacroAttribute on top of a method with a MacroCallInfo as a parameter, you would successfully create your own macro.

10.1 Macro Name

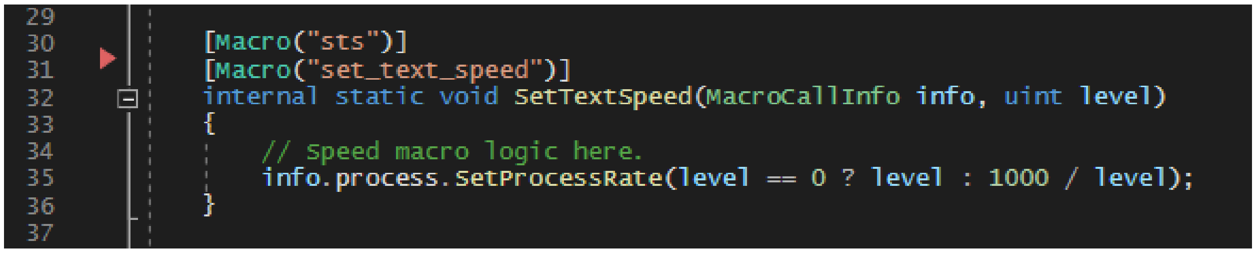
The name of the macro to successfully invoke it’s corresponding method.

10.2 Macro Endpoints

Macro Endpoints is another way of saying the macro’s associated method. This is the method to be called with Skriptr (within XVNML) sees a macro block containing that name.

10.3 Custom Macos Example

Here’s an example of one of the Standard XVNML Macros provided by the XVNML SCLA. See more at 6. XVNML Standard Macros.



```
[Macro("sts")]
[Macro("set_text_speed")]
internal static void SetTextSpeed(MacroCallInfo info, uint level)
{
    // Speed macro logic here.
    info.process.SetProcessRate(level == 0 ? level : 1000 / level);
}
```

11. XVNML SCLA: Dialogue Writer

Exclusive to the XVNML SCLA, the Dialogue Writer is a static class capable of displaying dialogue in a type-writer fashion. This is ran by a separate thread of it’s own to be in tandem with other threads (such as seperating XVNML related callback from the Unity’s Main Thread).

11.1 CDPs

CDPs (or Concurrent Dialogue Processes) are what is used when utilizing the Dialogue Writer. A new process will be created for each data from a <dialogue> element that is set to a channel. If you want to simulate multple people talk (such as in a crowd), you would utilize CDPs for this purpose.

11.2 Dialogue Writer Callbacks

There are a total of 12 different callbacks that you can make for the Dialogue Writer.

Property Name	Description
OnLineStart	Called when a Skriptr Line is first being processed.
OnLineSubstringChange	Called every time the character index of the overall content to be displayed changes.
OnLinePause	Called on a Skriptr Line is finished processing.
OnNextLine	Called when the Dialogue Writer goes to the next Skriptr Line.
OnDialogueFinish	Called when all Skriptr Lines in a dialogue has been processed.
OnCastChange	Called when a cast member changed.
OnCastExpressionChange	Called when a cast member's portrait or expression changes.
OnCastVoiceChange	Called when a cast member's voice changes.
OnPrompt	Called when a prompt has been initiated to the user.
OnPromptResponse	Called when feedback is given to a prompt.
OnChannelBlock	Called when the processing of a channel is inactive.
OnChannelUnblock	Called when the processing of a channel becomes active.

11.3 "AllocateChannels" Method

By default, there are up to 12 channels that you can have multiple dialogue processes run (and they are all controlled by the Dialogue Writer individually). You can use this method to only allocate one channel for one set of dialogue (which is perhaps the most common thing to do).

11.4 “SetThreadInterval” method

Change the update rate of the Dialogue Writer. The default thread rate is 1/100 of a second (10ms per update). If you are concerned of this value, or you are having issues with text being displayed by the Dialogue Writer, you can play around and change this value using this method.

11.4 “Write” Method

This method will take a Script to run that from a Dialogue element inside your XVNML file. Upon creating an instance of the XVNML dom, you can use the “GetElement” method to grab a dialogue that you would like to be processed.

11.5 “MoveNextLine” Method

Using the callback for a line finishing in a set of dialogue, you can tell the DialogueWriter to move to the next avaliable line by using this method.

11.6 “Block” and “Unblock” Methods

If you want to implement a pausing mechanism in (for example, a game), you can use the “Block” and “Unblock” method to control the flow of the line being displayed in a dialogue.

11.7 “ShutDown” Method

This is an important method to always consider, since you don’t want the DialogueWriter to run after an application has been closed. This method will invoke the CancellationToken and close the thread that the DialogueWriter is running on.

11.8 Usage

Here is a general usage of how to utilize the DialogueWriter. This example is done on a C# Console Application project.

```
// See https://aka.ms/new-console-template for more information
using System.Text;
using XVNML.Core.Dialogue;
using XVNML.Utilities.Dialogue;
using XVNML.XVNMLUtility;
using XVNML.XVNMLUtility.Tags;

static class Program
{
    private static bool finished = false;
    static void Main(string[] args)
    {
        XVNMLObj.Create(@"../../../XVNMLFiles/consoleApp.main.xvnml", dom =>
        {
            if (dom == null) return;
            if (dom.Root == null) return;

            Console.OutputEncoding = Encoding.UTF8;

            DialogueScript script =
dom.Root.GetElement<Dialogue>("MainTest")?.dialogueOutput!;

            DialogueWriter.AllocateChannels(1);
            DialogueWriter.OnPrompt![0] += DisplayPrompts;
            DialogueWriter.OnPromptResonse![0] += RespondToPrompt;
            DialogueWriter.OnLineSubstringChange![0] += UpdateConsole;
            DialogueWriter.OnNextLine![0] += ClearConsole;
            DialogueWriter.OnLinePause![0] += MoveNext;
            DialogueWriter.OnDialogueFinish![0] += Finish;
            DialogueWriter.Write(script);
            Console.Clear();
        });

        while (finished == false)
            continue;
    }
}
```

12. XVNML SCLA: Dialogue Writer Processors

Dialogue Writer Process are individual units of dialogue being ran on one of the DialogueWriter's channels.

12.1 On Process Initiation

Upon calling the "Write" method from the Dialogue Writer and passing some the output of a dialogue, it'll go through each Skriptr Line stored in memory, and figure out the general flow of each (accounting for lines nested in a prompt response). This will also the user to jump from and to a line without disruption of the Dialogue Writer.

12.2 “JumpToStartingLineFromResponse” Method

This method will tell the Dialogue Writer to go to the pre-defined index of a line based on what response the user has selected.

12.3 “JumpToReturningLineFromResponse” Method

After the end of the response (denoting by a Skriptr Line ending in the terminal character “<<”), the process will tell the Dialogue Writer to jump out of scope (depending on though the prompt and prompt responses where structured).

12.4 “LeadTo” Method

This method will tell the Dialogue Writer to the next or previous Skriptr Line (disregarding lines that are in or our of scope of a prompt response).

12.5 “JumpTo” Method

This method will tell the Dialogue Writer to jump to a specific index of a Skriptr Line, or by the name of a Skriptr Line. Named Lines are marks by square brackets.

12.6 “FetchPrompts” Method

Upon the end of a Skriptr Line in an Interrogative Role, it’ll get the SkriptrLine data of any Prompt Responses that it has, as well as their indices. Each data contains info on where to start if the response has been selected, and where to end after ending the response. This is all predetermined by the Skriptr Parser during the Tokenization and Parsing process.

12.7 Usage

Here is an example of using these methods (going off of the previous code exapmle from the Dialogue Writer):

```
// See https://aka.ms/new-console-template for more information
using System.Text;
using XVNML.Core.Dialogue;
using XVNML.Utilities.Dialogue;
using XVNML.XVNMLUtility;
using XVNML.XVNMLUtility.Tags;

static class Program
{
    private static bool finished = false;
    static void Main(string[] args)
    {
        XVNMLObj.Create(@"../../../XVNMLFiles/consoleApp.main.xvnml", dom =>
        {
            if (dom == null) return;
            if (dom.Root == null) return;

            Console.OutputEncoding = Encoding.UTF8;

            DialogueScript script =
dom.Root.GetElement<Dialogue>("MainTest")?.dialogueOutput!;

            DialogueWriter.AllocateChannels(1);
            DialogueWriter.OnPrompt![0] += DisplayPrompts;
            DialogueWriter.OnPromptResonse![0] += RespondToPrompt;
            DialogueWriter.OnLineSubstringChange![0] += UpdateConsole;
            DialogueWriter.OnNextLine![0] += ClearConsole;
            DialogueWriter.OnLinePause![0] += MoveNext;
            DialogueWriter.OnDialogueFinish![0] += Finish;
            DialogueWriter.Write(script);
            Console.Clear();
        });

        while (finished == false)
            continue;
    }

    private static void DisplayPrompts(DialogueWriterProcessor sender)
    {
        // We need to somehow graph the current line's prompt answers,
        // as well as where the process to hope to in response.
        var prompts = sender.FetchPrompts();
        Console.Write('\n');
        List<string> responses = prompts!.Keys.ToList();
        for(int i = 0; i < responses.Count; i++)
        {
            var promptData = responses[i];
            Console.Write($"{i}) {promptData}\n");
        }

        var response = Console.ReadLine();
        var responseIndex = Convert.ToInt32(response);
        if (responseIndex > responses.Count - 1) return;
        sender.JumpToStartingLineFromResponse(responses[Convert.ToInt32(response)]);
    }

    private static void RespondToPrompt(DialogueWriterProcessor sender)
    {
        // This is the point where we take the answer of our prompt
        // and tell the process to hope to a specified index of DialogueLines that
it has.
        // This luckily will increment as normal, because jumping to a dialogue line
is already
        // simple with predetermined indexes to go to.
        ClearConsole(sender);
    }

    private static void MoveNext(DialogueWriterProcessor sender)
    {
        if (sender.IsPass)
        {
            DialogueWriter.MoveNextLine(sender);
            return;
        }

        Console.Write("^");
        Console.ReadKey();
        DialogueWriter.MoveNextLine(sender);
    }

    private static void Finish(DialogueWriterProcessor sender)
    {
        DialogueWriter.ShutDown();
        finished = true;
        return;
    }

    private static void ClearConsole(DialogueWriterProcessor sender)
    {
        Console.Clear();
    }

    private static void UpdateConsole(DialogueWriterProcessor sender)
    {
        Console.SetCursorPosition(0, 0);
        Console.Write(sender.DisplayingContent);
    }
}
```