Getting Started

Table of Contents

Introduction

Welcome to CLARION! This guide is intended to help you get started working with the CLARION Library so that you can create and run your own task simulations using the CLARION cognitive architecture as your foundation. CLARION is a modern, true hybrid cognitive architecture that has been under development for many years by Dr. Ron Sun. Version 6.1 of the CLARION Library is the latest version, and is significantly more powerful and easier to use than its predecessors.

To get the most out of this guide, you should already have a basic knowledge of the CLARION theory and of cognitive modeling in general. Ideally, you should also be somewhat familiar with an object-oriented programming language (like Java, C++, or C#), and with object oriented programming (OO) in general. The CLARION Library, version 6.1, is written in C#, however if you are used to Java, do not be alarmed, as the languages are actually quite similar. If you already know Java, learning the minimal amount of C# that you need will not be very difficult.

In order to help you get familiar with C# and object-oriented programming we have provided links (like the one above), throughout this guide, to various resources on the web that we think may be helpful for gaining some extra background on certain topics and concepts that we will be discussing. So keep your eyes out for those links and feel free to click on them if you feel you need clarification on something.

Also, while we are certainly not in the business of advertising, if you are looking for a good reference manual on C#, we would highly recommend "C# [4.0] in a Nutshell" from O'Reilly Books.

Why C#?

Beginning with version 6.1, the CLARION Library is now written entirely in C#, while all previous versions were written in Java. "Why the switch to C#?", you may ask. Well, we had several reasons for abandoning Java and switching to C#. First, while both languages are equally "expressive," we found it much easier to build all of the new features and capabilities for version 6.1 of the CLARION Library in C#. This is owing to several language constructs in which Java is either lacking or (in our opinion) less proficient. Furthermore, C# provides many enhancements over Java, such as delegates, anonymous (lambda) functions, dynamic binding, enhanced event mechanisms, simple and straight-forward serialization, and LINQ. These feature (among others) have had the effect of greatly simplifying several aspects of the CLARION Library and we are confident that once you begin working with it, you will agree.

We should also note that, while it was originally developed by Microsoft, C# was and is intended to be an open language with both ECMA and ISO standards behind it. So those of you who tend to be afraid of that "evil monopoly" should rest easy. C# can be developed perfectly well on other operating systems by making use of the open-source (and free) Mono project development environment (Mono Develop). We will also point out (again, we are not in the business of advertising here) that if you prefer developing in Windows, Microsoft also provides a free version of their software (Visual Studio Express). The CLARION Library has been tested using both of these development environments, so please feel free to use whichever you prefer.

At this time, we won't go any further into discussing why we chose C# over Java (or a different language for that matter). The debate between the "superiority" of various programming languages (and styles) is ongoing and eternal. It would not serve the purpose of this document for us to spend any more time on discussing the benefits, differences, and/or shortcomings of C# versus Java. However, for those of you who are interested in further reading, a very thorough breakdown between C# and Java can be found online on Wikipedia.

We hope, however, that you are at least intrigued enough to want to try out this new system, so let's jump into it.

Installation

To begin, you need to know that the new CLARION Library leverages many of the features specific to version 4.0 of the .NET framework. Therefore, regardless of which development environment you use, it is import to make sure that you have the most recent .NET (or Mono) framework installed on your machine. If you are using Windows XP (service pack 2) or later, the newest .NET framework should be

downloadable through Windows update (if it is has not been automatically installed already). Otherwise, the mono framework can be found here.[1]

Installing either of the development environments that we have been discussing so far is fairly straightforward and instructions can easily be found on their websites, so we won't get into the details for installing these development environments here. Instead, we are just going to assume at this point that you have successfully installed the appropriate framework and development environment and are ready to begin.

About the CLARION Library package

To start, since you are reading this document, we can assume that you have already downloaded the zip file for the CLARION Library and have unzipped it to a folder somewhere on your hard drive. Within this folder you will find several things along with this guide. Specifically, you should note the following contents:

- The Getting Started guide (this document)
- The CLARION Library dynamic link library (dll)
    - The "assembly" (i.e., resource library)
- CLARION Library IntelliSense XML file
- A "Tutorials" folder
    - Contains guides (like this one) for configuring and customizing various aspects of the CLARION Library
- A "Samples" folder
    - Contains basic, intermediate, and advanced simulation samples
- A "Documentation" folder
    - Contains "MSDN-like" API reference documentation for the library

While we have striven, in this implementation, to simplify the process as much as possible, there can still be a bit of a learning curve when you are first starting with the CLARION library (and theory for that matter). Therefore, we encourage you to peruse the various documents, tutorials, and samples that have been included with the CLARION library before you begin using it. These items are provided in order to help clarify any confusion that may arise while you are developing your simulations. Additionally, taking the time to peruse the tutorials first should help minimize the amount of time it takes you to become a competent builder of CLARION-based agents.

Let's turn our attention now to setting up a simulation. We will also go over a few key concepts that you should keep in mind when using the CLARION Library.

[1] The individual implementations for the mono framework and development environment vary somewhat based on the operating system. Therefore, you may need to do some separate searching online to find the latest builds of mono for your OS.

Creating a Simulation Project

The first thing you need to do when setting up a simulation is to specify a "solution" for your task simulations. If you have not already created one, you should do so now. The specifics on how to do this varies slightly based on the development environment you are using, so please consult the guides for your particular environment if you need help creating a new solution. Feel free to call your solution whatever you would like (we suggest calling it "CLARION Simulations").

After we have created the solution, the next step is to create a new project (again, please consult the guides for your particular development environment if you need help with how to do this). A solution can house many different projects, so you do not need to create a new solution for every simulation you are going to write. However, each simulation should be in its own project. Creating a new project can usually be done within the "solution explorer" by simply right-clicking on your solution and choosing the "new project" option under the "add" menu item. Again, feel free to name the project whatever you would prefer. We recommend that you name your project so that it succinctly describes your simulation or task (e.g., "*SimpleHelloWorld*", as will be demonstrated later in the "*Setting Up & Using the ACS*" tutorial).

Referencing the CLARION Library

If you are somewhat unfamiliar with programming, we will need to spend a few moments discussing the concept of a "resource" (if you are familiar with it then feel free to skip this part). A resource is a collection (i.e., assembly or library) of objects (classes, interfaces, delegates, etc.) that you can use within your own code. The CLARION Library is simply a collection of this nature. In other words, the CLARION Library is not, in and of itself, executable. Instead, it provides you with the necessary tools in order to build CLARION-based "agents" within your own executable simulating environment. In fact, the files located in the "Samples" folder all serve as examples of this sort.

To use the CLARION Library in our project, we must add it as a resource to our project. Accomplishing this tends to vary based on the development environment, so you should consult the guides for your particular one if you need help with how to do this. However, in general, the process usually involves something like the following:

- Under your project (in the solution explorer), there is a "folder" named something like "resources" (or possibly "references"). Right-click on that folder and choose the "add" menu item from the drop-down.
- In the window that comes up, navigate to the location where you unzipped the CLARION Library, and select the "CLARIONLibrary.dll" assembly file.

Once you have completed these steps, the CLARION Library should appear in the "resources" (or "references") section under your project in the solution explorer. If it is listed there, then you have successfully specified the CLARION Library for your project, and will be able to use it.

Now that you have the CLARION Library resource referenced, lets look at how the library is structured.

The CLARION Library Namespaces

Clarion				
Clarion.Framework			Clarion.Plugins	Clarion.Samples
Clarion.Framework.Core	Clarion.Framework.Extensions[3]	Clarion.Framework.Templates		

The table above provides you with a hierarchal breakdown of the namespaces for the CLARION Library. You can consult the API reference document to get a complete list of all the classes within each namespace. However, in general, the namespaces are organized as follows:

- **`Clarion`** – This is the base namespace for the CLARION Library and it contains three classes: `World`, `AgentInitializer`, & `ImplicitComponentInitializer`.
- **`Clarion.Framework`** – This namespace contains the majority of the classes needed for initializing and running a simulation. Most of the classes contained within this namespace can be correlated directly to terms or concepts from the CLARION theory.
- **`Clarion.Framework.Core`** – This namespace contains the necessary constructs for the core operation of the system.[2]
- **`Clarion.Framework.Extensions`**[3] – This namespace contains extensions (e.g., meta-cognitive modules, various components, etc.) for the CLARION Library that, while not (necessarily) being specified within the CLARION theory itself, can still be used within an agent in the same fashion as the classes found in the `Clarion.Framework` namespace.
- **`Clarion.Framework.Templates`** – This namespace contains abstract classes, interfaces, and delegates, etc. that act as "templates" for building some custom user-defined objects (e.g., implicit components, drives, etc.).
- **`Clarion.Plugins`** – This namespace provides various plugins and tools that may be used in by a simulating environment to enhance the capabilities and applications of CLARION-based agents setup using the CLARION Library.
- **`Clarion.Samples`** – This namespace contains several simulating environment examples that serve as guides to help you learn how to use the CLARION Library.

---

[2] You should rarely need to access this namespace in order to initialize or run an agent.
[3] Note that an additional namespace, ***Clarion.Framework.Extensions.Templates***, has been added, which provides some useful templates for creating extensions

We should note that the `Clarion.Samples` namespace is not actually in the assembly (i.e., dll file). Instead, the files that constitute this namespace are located in the "Samples" folder. These samples have been placed in the "`Samples`" namespace because they are provided as part of the overall CLARION Library package (i.e., zip file), and thus can be considered as being a tangential part of the library.

Basic Guideline for Setting Up Simulations

First, we should stress that there are **NO** "requirements" for setting up a simulation (although there are certainly some requirements for setting up an agent). If you have any prior experience with developing simulations using cognitive architectures, you are probably aware of just how quickly it can become very difficult to maintain a strict format for developing simulations, especially as they increase in complexity. With this in mind, in the CLARION Library, we have resisted imposing rigid guidelines when creating simulating environments. Instead, we provide a general outline for setting up and running a simulation. The following describes the approximate steps you will usually want to take:

1st. Describe the features and objects in the world

2nd. Define the actions and motivations (goals) that will dictate how the agents interact with the world

3rd. Initialize the agents' internal functions so that they can make action decisions based on how they perceive the world

4th. Provide mechanisms to enable the agents to interact with the world

Keep this guideline in mind when you are developing your simulating environment. Maintaining this approximate structure will help reduce the amount of time you might need to spend "debugging" your simulation once it is written.

At this point, we will conclude our introduction by talking about an important concept. That is, the distinction between two key aspects of the CLARION Library: "descriptive" and "functional" objects.

Descriptive vs. Functional Objects

Let's begin by talking about descriptive objects. Descriptive objects are objects that are used to describe the features of things (e.g., the simulating environment, agents, actions, goals, declarative knowledge, etc.). In other words, they are interested in how the simulating environment looks. They include things like:

- Dimension-value pairs
- Agents
- Chunks (actions, goals, etc.)

In essence, a descriptive object does exactly what it suggests: it describes things. Within the CLARION Library, descriptive objects are generated, stored, and retrieved exclusively through the `World` singleton object. The `World` is a so-called "singleton" object created for you by the CLARION library. It already exists by the time your code starts running, so it is always available for you to access. Also, it is important to know that all of your interactions with the `World` object are done statically. In other words, making calls to the methods of the "world" is done directly through the class name. For example, suppose you want to specify the dimension-value pair `{Dim1, Val1}` as a feature of the world. The following line of code demonstrates how this would be accomplished:

```
DimensionValuePair dv1 = World.NewDimensionValuePair("Dim1", "Val1");
```

Thinking about it conceptually, the `World` essentially contains "everything":

- The entire environment of your task
- The agents that exist within the environment
- Any special "internal" information that pertains to those agents[4]

There are plenty of more details regarding descriptive objects, but we will get into those other considerations in later tutorials. However, you should at least grasp the concept by this point, so we'll move over to the functional considerations of the library.

A functional object is an object within the CLARION Library that actually does something. In other words, functional objects are used to describe the processes and mechanism that make up the inter-workings of an agent. In general, we refer to functional objects as "components" or "modules." These components and modules include the following:

- Implicit Decision Networks
- Action Rules
- Drives
- Meta-cognitive Modules
- Etc.

Notice that the above names refer less to the actual classes that make up the various functional objects provided by the CLARION Library and instead represent the location (or container) in which the functional components are stored within the agent. These "internal locations" are referred to as the agent's `InternalContainers`[5] within the library. We will get into the specifics of this a bit

[4] The concept of "agent-specific internal information" is beyond the scope of this guide. Feel free to consult the "*Intermediate Tutorials*" for more details concerning this topic.
[5] Defined by an enumerator of the same name located within the `Agent` class

more in the later tutorials.[6] For now, just be aware that the functional objects are generated using the static "initialize" methods that located in the `AgentInitializer` class.

By this point, you now have the necessary foundation to get started with building simulations using the CLARION Library. In the next tutorial, we will walk you through a simple example of how to setup a simulating environment, initialize an agent, and have that agent perform a task within the environment. This walk through can be found in the "*Setting Up & Using the ACS*" tutorial, which is located in the "*Basic Tutorials"* section of the "*Tutorials*" folder.

Also, be aware that if you run into any problems, have additional questions, or want to report a bug, you can contact us at clarion.support@gmail.com.

[6] Specifically, see the "*Useful Features*" guide located in the "*Features & Plugins*" section of the "*Tutorials*" folders