Gateway Developers' Toolkits

products to allow others to extend Habitat™

Chip Morningstar

Lucasfilm Ltd. Games Division
September 25, 1986

Introduction

What we are calling a *Gateway Developer's Toolkit* is a package of software and documentation to allow third-party developers to develop new products based on the *Habitat* system. As I see it, there are actually three levels of involvement that we may wish to support. These are:

- Realm Developer's Toolkit
- World Developer's Toolkit
- Universe Developer's Toolkit

I'll talk about each of these in turn.

Realm Developer's Toolkit

Description

The *Realm Developer's Toolkit* is a package to enable the creation of new realms for the existing *Habitat* world. Realm generation involves the following activities:

- Design, definition, and generation of regions.
- Definition and assignment of inter-region connectivity and topology.
- Definition, creation, and placement of objects (using an existing object set).
- Definition and distribution of ''printed matter'' (books, notes, signs, etc.).
- Definition of various ancillary databases (e.g., TelePort addresses).

Essentially, realm generation requires the developer to create a set of entries for each of the various host system *Habitat* databases. Collectively, these database entries describe everything about a set of regions and their contents.

Components

The toolkit itself consists of the following components:

- *Region Editor* — allows the developer to generate region definitions. A region is defined by its contents: a set of objects and their properties. The Region Editor therefor allows the developer to create and delete objects, and to modify their state information, such as location, color, graphic state, and so on. It displays the composite image that results from putting all these objects together, so that the developer can see what he has wrought.
- *Realm Editor* — allows the developer to generate realm definitions. A realm is defined as a set of regions. The things that the Realm Editor manipulates on the developer's behalf are the interconnectivity of the various regions and their orientations with respect to each other. In an ideal world, the Region Editor and the Realm Editor would be separate modes of a single, integrated utility, allowing the designer to pop back and forth between levels of detail quickly and easily.

- *Database File Translator* — a utility that converts the various outputs of the region editor, the realm editor, and other random text files into a single, uniform format that can be read and internalized by the Genesis software on the host computer. It is used both to package the data for shipment to Quantum and to integrate various other, simpler pieces of data with the whole. These other pieces of data include text for documents to be found in the realm, lists of TelePort addresses, and so on. As with the Realm Editor, ideally this would like to be part of an integrated package that subsumes the other utilities.
- *Realm Developer's Guidebook* — the documentation for the above described utilities together with a complete guide to the practice of realm development. This should include various ''tricks of the trade'', rules of thumb, and general guidelines for making a realm that is interesting and self-consistent.
- *Object Manuals* — one for each object disk that we support. Each is a reference guide that describes each of the objects on the disk: what it looks like, what it does, what its state variables are and how they are used, and what sorts of things the object can be used for.

Current Status

The current incarnation of the Region Editor runs on the Commodore 64 but requires Fastlink. The simplest and most obvious extension is to make it run as a standalone utility. This would require the addition of a number of functions which are currently achieved by poking around the machine with Fastlink (such as the creation of new objects), and a means of saving the results to (and reading them back in from) a Commodore disk file. Also, the Region Editor currently works from a regular *Habitat* object disk, so the toolkit would have to include the regular *Habitat* software in addition to its own specialized components.

Realm editing is currently achieved by manually editing a text file using Emacs. This text file is actually a Unix shell-script that directs a utility called ''Slur'' to convert the Region Editor output files into a format suitable for input to another program called ''Riddle''. Riddle in turn generates files suitable for transmission to Quantum and subsequent processing by Genesis. In addition to requiring a number of specialized, local utility programs, it is all a very tedious, cumbersome, and error prone process.

Product Directions

It is unclear at this point what machine the software components should run on. Our current development environment is a Commodore 64 with a permanent Fastlink umbilical connecting it to a Sun Workstation running Unix. This is obviously not the target system we want to address, even if the community of users is very, very small, since nobody else has Fastlink and almost nobody (in our business) has a Sun.

The Realm Developer's Toolkit is a product that we might wish to see widely distributed, to allow those users with a bent for this sort of thing to become involved in the world generation process. Since new realms can be plugged into the existing world with relative ease, the *Habitat* system could become a publishing medium for users' realm designs with ourselves (i.e., Quantum and Lucasfilm) as editors. If this is the case, then the most plausible host for the Realm Developer's Toolkit is the Commodore 64 itself.

Since the current Region Editor is largely Commodore 64 based, we already have a good starting point. As mentioned above, the extensions to the Region Editor required to make it a standalone system are obvious, though not trivial. The Region Editor, in fact, really should be Commodore 64 based, since the ultimate decision that needs to be made is whether or not a given region looks right when displayed on a Commodore 64 screen. To base the entire Realm Developer's Toolkit on the Commodore 64 would require some considerable effort, however, since the other tools that make up the package would have to be written from scratch.

There are off-the-shelf products (such as Kermit) that enable the transfer of files between Commodore 64s and other machines via regular terminal lines. One approach to follow, therefor, would be to base the Region Editor on the Commodore 64, but leave the remaining tools on a more substantial host (such as our Sun systems). Since these tools, such as they are, are written in **C**, they are easily ported from one Unix system to another. This means that our tools could run on any Unix host, even if it is not a Sun. Since the tools will have to be cleaned up and enhanced anyway, we could re-code them with an eye to system independence as well, so that they could (in theory at least) run on any machine that had a **C** compiler

(such as an IBM PC/clone, a Macintosh, an Amiga or an Atari ST). Depending on how free we wish to be with these tools, we could simply distribute the source code via the usual channels (Usenet, users' groups, and so on) and let the people out there in the world worry about adaptation of the tools to a wide variety of machines. On the one hand, tightly holding onto the tools' source enables us to retain control over distribution and to use sale of the Toolkits as a revenue source. On the other hand, distributing the tools freely encourages the proliferation of efforts to develop products for our system and also helps establish our system as a ''standard''.

World Developer's Toolkit

Description

The *World Developer's Toolkit* is a package to enable the creation of new object sets that will function within the existing *Habitat* universe. This will enable the creation of new worlds with completely different imagery and game play but which build on top of the existing software foundation. In particular, these new worlds will use the same main *Habitat* program and the same host region processor. They replace the Commodore 64 object disk and the host object behavior routines. World generation involves the following activities:

- □ Design of new artwork and animation.
- □ Design of new sound effects.
- □ Programming of new object behaviors.
- □ Integration of newly generated elements into a complete Commodore 64 object disk.
- □ Generation of new realms utilizing the new object set.

Essentially, world generation requires the developer to create a set of entries for each of the various Commodore 64 system *Habitat* databases that reside on the object disk, and then to create one or more realms to match. Since each object also has a software component that reside on the host, programming the host computer is also required.

Components

The World Developer's Toolkit subsumes the Realm Developer's Toolkit, and adds a package of utilities and information for the development of new worlds. Creation of a new world, in essence, consists of the definition of a new object set and a collection of realms that use it. In addition to the Realm Developer's Toolkit components, the World Developer's Toolkit contains:

- □ *Animation Cel Editor* — a utility that allows the developer to create cel images for objects and avatars. It is essentially a graphics editor with a built-in animation capability that mirrors that found in the *Habitat* main program.
- □ *Graphics Definition Utility* — allows the graphics associated with an object to be defined. This combines the output of the Animation Cel Editor with other graphics primitives that our graphics system supports to define the complete set of imagery for each object.
- □ *Sound Effects Editor* — a utility that allows the developer to create sound effects for objects. It is essentially an interactive front-end for the *Habitat* sound driver that lets the user manipulate the contents of a sound effect definition directly.
- □ *Sound Definition Utility* — analagous to the Graphics Definition Utility, it allows the sound effects created with the Sound Effects Editor to be integrated with objects and with each other.
- □ *Macro Assembler* — just an ordinary assembler. Ideally this need not be part of the package, but since we are using a special set of assembly-language macros to realize the object behavior code, we may need to provide a matching assembler.
- □ *Object Actions Macro Package* — a special set of assembly language macros used in the coding of object behaviors.
- □ *Muddle* — a utility that takes the various images, sound effects and fragments of object behavior code and generates the tables and files that go on a *Habitat* object disk.

- *PL/1 Object Library Package* — a set of PL/1 subroutines and include files that are used to code the host-resident portion of the object behaviors.
- *World Developer's Guidebook* — the complete set of documentation for all of these components.

Current Status

The current Animation Cel Editor is yet another Commodore 64/Sun/Fastlink hybrid. The final output of the editor is a Macross source file that defines the data tables for a given set of cels. The functions of the Graphics Definition Utility are currently realized by manually editting these tables. Sound effects are handled similarly.

Object behaviors are coded in 6502 assembly language using Macross using a special set of macros which are already coded.

The Muddle utility runs on a Sun workstation. It reads a special file that describes the layout of the object disk components (images, sounds, and actions). It reads these from the various files they reside in, computes a set of tables, and then outputs a series of data files containing the tables and the object components themselves. These data files are downloaded to the Commodore 64 disk using Fastlink.

Host object behaviors are coded in PL/1 using the conventional Stratus software development tools. They are then compiled and bound directly into the region processor program. Each world would therefor have to have a separate region processor that combined the generic region processing part of the program with a set of world-specific object behavior routines.

Product Directions

Most of the comments made above about the Realm Developer's Toolkit apply here as well. As with the Region Editor, the Animation Cel Editor and the Sound Effects Editor need to be adapted to operate in a standalone mode on the Commodore 64.

When developers are creating new objects, they need to program simultaneously on two fronts: in assembly language on the Commodore 64 and in PL/1 on the Stratus.

For programming the Commodore 64, we could, conceivably, distribute Macross. However, it is not likely to port easily to non-Unix systems since it absolutely demands a virtual-memory system on which to run. We could produce a document describing the macros we use so that they could be adapted (by others) to a variety of other development systems on other machines — we don't do anything too bizarre with Macross macros that could not be adapted to another assembly language.

Programming the Stratus host can proceed as it does now, using the conventional Stratus tools. Obviously, access to the World Developer's Toolkit will have to be on a much more restricted basis than the Realm Developer's Toolkit, since it requires access to the host computer at a fairly intimate level. This means that world developers would have to be carefully screened and that any world development efforts could only take place within the enclosure of appropriate, detailed legal agreements between Quantum, ourselves, and the developers.

Universe Developer's Toolkit

Description

The *Universe Developer's Toolkit* is a package to enable the creation of entirely new systems based on some or all of the underlying *Habitat* structural components. Universe generation involves the creation of a complete system that does not necessarily have to be compatible with the existing *Habitat* system in any way. Components of the existing system are simply used to reduce the amount of work required and to minimize ''reinventing the wheel''.

Current Components

These are the components of the existing system that we might wish to break out into separate building blocks:

- □ Commodore 64 RS232 port driver.
- □ Commodore 64 Q-Link message protocol handler.
- □ Commodore 64 Habitat object/message protocol handler.
- □ Commodore 64 Habitat user interface.
- □ Commodore 64 animation engine.
- □ Commodore 64 fast, interruptable disk I/O routines.
- □ Commodore 64 sound effects driver.
- □ Commodore 64 object database and memory manager.
- □ Host message I/O routines.
- □ Host region processor.
- □ Host object database engine.

The Commodore 64 software components are all written in the 6502 assembly language Macross. The host software components are all written in Stratus PL/1 and assume they are running in the Q-Link runtime environment. In addition, the Universe Developer's Toolkit should also include the tools and utilities of the World Developer's Toolkit and whatever other Unix, Commodore 64, and Stratus tools and utilities we care to throw in.

Product Directions

All of the issues discussed above for the World Developer's Toolkit apply to the Universe Developer's Toolkit as well. In addition, it would be difficult to separate our 6502 software components from the Macross environment, though certainly not impossible. The inclusion of Macross as part of the Universe Developer's Toolkit is probably desirable, which in turn requires that universe developers be restricted to a fairly narrow range of potential development hosts. This is probably not too unreasonable, since universe development is such a major undertaking that any plausible developer could be expected to either have or obtain a Sun-class development host or invest the effort to convert the Commodore 64 code to their own development environment. Alternatively, they could work very closely with us and actually use our development system, though this would no doubt require complex negotiations.

Obviously, no universe development effort could take place without detailed organization arrangements being set up in advance. The nature of the activity is such that it is not practical to simply sell a software package and then abandon the developers to their own devices, nor would we care to have the products at this level slip out of our control.