Gateway Development

products and activities that extend Habitat™

Chip Morningstar

Lucasfilm Ltd. Games Division
October 13, 1986

Introduction

What we have been 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 to consider. These are:

- □ Realm Development
- □ World Development
- □ Universe Development

I'll talk about each of these in turn.

Realm Development

A *realm*, in our terminology, is a collection of *Habitat* regions sharing a common theme and purpose. *Realm development* is the creation of new realms for use within the existing *Habitat* world. The*Realm Developer's Toolkit* will be a package to enable third parties to do this. New realms use the existing *Habitat* software (both host and Commodore 64) and an existing *Habitat* data disk.

Realm development 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).

Realm development 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.

Realm developers will create new realms using the *Realm Developer's Toolkit*. This is a package of software and documentation that makes it possible for them to generate the appropriate files, suitable for uploading to QuantumLink. The toolkit consists of a special editor program, the *Realm Editor*, an ordinary *Habitat* object data disk, and a thick manual, the *Realm Developer's Guidebook*.

The *Realm Editor* is an editor program that runs on the Commodore 64. It generates, on a Commodore 64 diskette, a standard set of data files in a standard format. These files collectively describe a realm in all the specificity that the host requires to generate the correct set of *Habitat* database entries for it. The Realm Editor is dual-moded.

In *Realm Mode* it allows the developer to manipulate the global properties of a realm — what regions it contains, their interconnectivity and their orientations with respect to each other.

In *Region Mode* it allows the developer to manipulate the individual regions themselves. A region is defined by its contents: a set of objects and their properties. The developer can create and delete these objects, and modify their state information, such as location, color, graphic state, and so on. The editor

displays the composite image that results from putting all these objects together, so that the developer can see what he has wrought.

Once the developer has created a realm definition using the Realm Editor, he must transfer his creation to the host system so that it can be installed in the world. This is accomplished by uploading the files that were output by the Realm Editor, together with any additional supporting material, such as text for books and documentation for the system operators (these text files may be created with whatever conventional Commodore 64 text-editing program the developer cares to use). A special program on the host, called the *Realm Database File Translator*, converts the uploaded files into the appropriate host database entries, which can then be examined, tested, or installed in the running system.

The *Realm Developer's Guidebook* is the developer's bible, documenting both the Realm Editor itself and the whole process of realm development. This will include various ''tricks of the trade'', rules of thumb, and general guidelines for making a realm that is interesting and self-consistent. As part of the *Guidebook*, there will be one or more *Object Manuals* — one for each object disk that we support (initially, of course, there will be only one of these). An *Object Manual* is a reference guide that describes each of the objects on an object 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.

The current incarnation of the Realm Editor runs on a Commodore 64 connected via Fastlink™ to a Sun Workstation. Most of the region editing functions are performed by the Commodore 64, but the realm editing functions, as well as long-term file storage, are handled by the Sun. To make it run as a standalone utility will require the addition of a few Region Mode functions (such as the creation of new objects) which are currently achieved by poking around the machine with Fastlink, and a means of saving the results to (and reading them back in from) a Commodore disk file. The Realm Mode functions will also have to be coded. Fortunately, we can build on top of the existing components for most of this. In addition, since it must work from a regular *Habitat* object disk, we must add the appropriate interactive functions to handle disk swapping as needed.

Realm definitions currently pass through a variety of Unix utilities that eventually produce files suitable for transmission to Quantum. These files are passed through a utility named *Genesis* that runs on the Stratus, producing the actual database entries. The file format, though readily manipulated by both Unix- and Stratus-resident utility programs, is not suitable for use in the Realm Developer's Toolkit package because it is far too bulky for implementation on a Commodore 64. This means that the *Realm Database File Translator* utility must still be written for the host system.

World Development

What we call *World Development* is the creation of new object sets that function within the existing *Habitat* universe. This in turn results in the creation of new worlds with completely different imagery and game play but which build on top of the present software foundation. In particular, these new worlds 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 development 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 development 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 resides on the host, programming the host computer is also required.

World development subsumes realm development, since any new object disk will require a new set of realms to go with it. Furthermore, the same set of software tools for realm generation may be applied to a new object disk without change. However, world development requires more effort and results in a more invasive change to the system since it *does* require additional programming (to implement the object

behaviors) both for the Commodore 64 and for the host, whereas realm development requires no special programming.

Given detailed specifications for the Commodore 64 disk database formats, the data formats and capabilities of the cel animation engine, the choreography engine and the sound effects driver, the specifics of both the host and the Commodore 64 object behavior execution environments, and the object-message communications protocol, it is conceivable that one could generate a new ''world'' without special assistance (a new world, in this context, is a functioning Commodore 64 object disk and a set of host object behavior routines to match). However, the complexity of this task would be somewhat daunting, since a developer would have to spend a while developing some specialized software tools. We have such specialized utilities for our development system, but they are not likely to port easily to other environments. These utilities include:

- *Animation Cel Editor* — a utility that enable the creation of 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.
- *Habitat Sound Effects Editor* — a utility that enables the creation of 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.
- *Macross* — a very powerful macro assembler, for which we have developed a set of macros to ease 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.

In addition we understand how the various pieces of the system fit together. In sum, the mechanisms of world creation are not readily transferred to other developer's, though it would not be impossible given copious documentation.

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 must therefor have a separate region processor that combines the generic region processing part of the program with a set of world-specific object behavior routines. Coding these routines also requires specialized knowledge, though it can be simplified by use of certain utility subroutines which we have developed. Since world development does require programming the host, world developers would have to be carefully screened and any world development efforts could only take place within the enclosure of appropriate, detailed legal agreements between Quantum, ourselves, and the developers.

Universe Development

By *Universe Development* we mean the creation of entirely new systems based on some or all of the underlying *Habitat* structural components. Universe development involves the creation of complete systems that do not necessarily have to be compatible with the existing *Habitat* system in any way (although they could be if we so desired). Components of the existing system would simply be used to reduce the amount of work required and to minimize ''reinventing the wheel''.

Commodore 64 software components of the existing system that could be useful in *Habitat*-technology spinoff products include the RS-232 port driver, Q-Link message protocol handler, Habitat object-message protocol handler, Habitat user interface routines, cel animation engine, high-speed interruptible disk I/O routines, sound effects driver, and object database/memory manager. Useful host software components include the message I/O routines, region processor and object database engine, as well as the entire support infrastructure of the broader Q-Link system.

While these are all useful building blocks, they do not lend themselves readily to being individually separated for distribution to other developers. Instead, they represented the cumulative expertise of the Lucasfilm and QuantumLink technical staffs in the implementation of systems of this type. In other words, it would be *very* difficult to make these components available to another developer, but they are readily available to us if we choose to implement additional systems based on *Habitat*, and they will considerably reduce the time and expense of developing such systems if we use them.