Terazona 1.1 to 1.2
Migration Guide

August 22, 2003

Table of Contents

Terazona 1.1 to 1.2 1
Migration Guide 1
Summary 3
New Installation Requirements for 1.2 3
New Additions for 1.2 4
Zona Modeler 4
Zona Modeler Architecture 5
Zona Modeler Inputs & Outputs 7
Zona Modeler Run Time 9
Zona Modeler Initialization 9
Zona Modeler Design Time 10
Zona Modeler User Interface 10
Zona Modeler Object Deployment 13
Game Guilds 15
Game Guild Structure 15
Game Guild Operations 16
Client API Changes 18
ZonaServices.h Additions 18
ZonaServices.h Changes 18
GameStateManager.h Deletions 18
GameStateCallback.h Changes 19
ZonaClientEntity.h Additions 19
ZonaClientEntity.h Deletions 19
ZonaClientCharacter.h Deletions 20
ZonaClientEntity.h Changes 20
Game Server API Changes 20
ZonaGSValidate.h Changes 20
Game Server API Additions 21
GameGuildServiceInterface.h Additions 21
GameGuildCallback.h Additions 22
ZonaBaseGuild.h Additions 23
ZonaBaseGuild.h Property Attributes 23
ZonaBaseGuild.h Inviter Attributes 24
ZonaClientGuildMembership.h Additions 24
ZonaBaseGuildMembership.h Additions 24
ZonaGuildChatMsg.h Additions 25

Summary

This document briefly describes the change made between the Terazona 1.1 Release and the 1.2 Release. There are several new core additions to the system as well as changes to the API. These are the significant changes in the Terazona 1.2 Release:

- **Zona Modeler** – an XML-based rapid application development (RAD) tool for creating bandwidth-optimized game objects within networked game architectures. Zona Modeler auto-generates Java and C++ code for easy Client- and Server-side integration and deployment.
- **Game Guilds** – Provide a set- or group-based approach to propagating Entity state updates within Terazona. Previous Entity State Management (ESM) update functions were based on spatial or local paradigms and enforced specific game play restrictions on game developers. Group-based Entity state updates provide new ways to enhance game playability and depth.
- **User Administration** –`ZonaPass.exe` has been deprecated. To create additional users, use the `%ZONA_HOME%\bin\createDemoUser.bat` (Windows) or `$ZONA_HOME/bin/createDemoUser` (Solaris/Linux) utility with usage as follows:

  ```
  CreateDemoUser <user_name> <password> [<user_role> [<count>]]
  ```

- **Auditing** – Game State Auditing requires significant modifications. This modification will not be completed for the Terazona 1.2.0 release. Therefore, Game State Auditing is unavailable until a forthcoming patch release. Chat Server Auditing still functions normally.
- **API Updates** – The C Application Program Interface (CAPI) and Game Server API (GSAPI) have been updated to provide new functionality, deprecate some outdated functionality, and to simplify some procedures. In particular, the old network object model consisted of:
  1. Public Display Properties
  2. Public State Properties
  3. Private Properties
  4. System Properties

- This has been simplified. The new model uses three properties:
  1. Public Properties
  2. Private Properties
  3. System Properties

New Installation Requirements for 1.2

Because of the integration of Zona Modeler into the Terazona system, Java J2SE 1.4.2 JDK must be installed on all development machines. Sun's licensing requirements do not allow Zona to bundle this with our Installer; you must download and deploy this package yourself. Run-time execution requires only the Java J2SE 1.4.2 JRE.

New Additions for 1.2

Zona Modeler

Zona Modeler provides a rapid and comprehensive solution to the problem of creating and optimizing networked game objects to communicate and update Entity states across distributed game environments.

Previously, creating such network objects required programmatic development of data structures (such as C++ `structs`) within the main body of game code. There were a number of problems with this approach.

Game logic was intermingled with network game object declaration. Furthermore, there was no central domain where network game objects could be defined and this created the potential for redundancy and mismatch between NPC Server and GSS game code.

Zona Modeler enables Terazona developers and designers to separate network game object development from game logic development. This decoupling also makes possible an enhanced workflow during design-time where game developers work on game logic and game designers work on game object design).

During run-time, Zona Modeler optimizes the bandwidth of the network messages required to propagate Entity state updates. Before Zona Modeler, to update a single property attribute of a game object, the Client would have sent a complete binary object ("blob") across the wires to the GSS. Each subscribed GSS would then have to extract the object from the blob, identify the changed parameter, validate the change, and update its server-side game objects.

Configuring Clients and GSSs to send only the changed values for specific attributes ("entity deltas") required the explicit setting and clearing of dirty, or changed, property attributes. Although this conserved bandwidth and reduced unnecessary messages, it was cumbersome.

Zona Modeler now handles all entity deltas transparently. Network game objects created or modified within the Zona Modeler graphical user interface (ZMUI) generate bandwidth-optimized C++ and Java code that can be compiled and deployed across the Terazona server cluster and on Clients.

Zona Modeler Architecture

Zona Modeler is focused on the creation and modification of network objects. Terazona games typically execute across large, possibly heterogeneous game server clusters. A key component of any such distributed system is describing an efficient method of communicating object state changes between different machines in the cluster.

This process of sending object states and object state changes between execution machines across the "wire" (or network layer) within a distributed system is called marshalling (for sending) and unmarshalling (for receiving). This is similar to the concept of serialization.

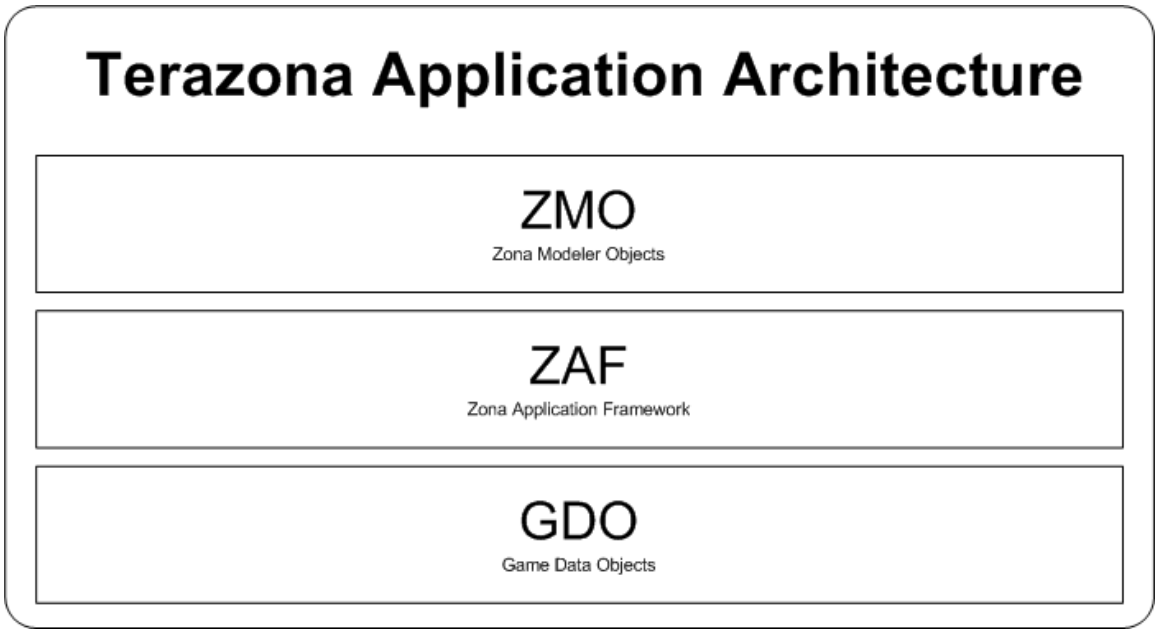
Within distributed systems, to perform marshalling and unmarshalling each execution machine requires various meta-object support data commonly known as interfaces, skeletons, and stubs. Zona Modeler creates these for you during design-time.

A Terazona application can now be represented by a 3-level architecture. The base level, ZMO, handles network game objects and is transparent to game developers and game designers. Terazona developers interact primarily with the middle layer, ZAF, that handles in-game object execution and management. The final layer, GDO, transparently handles in-game object instantiation.

Each data object layer talks to an associated layer-specific Controller object that provides the following services:

1. Object-Relational Mapping
2. Messaging
3. Validation

Zona Modeler abstracts the network game object layer below the Zona Application Framework (ZAF) game execution layer. To manipulate ZAF, you use the classic CAPI and GSAPI functions. Using the 3-layer model, the new Terazona class hierarchy looks like this:



There are three fundamental base classes used when interacting with ZAF:

1. ZonaBaseEntity – forms the basis for the classic ZAF Entities, such as ZonaClientEntities, ZonaServerEntities, Chat Guilds, and so on.
2. ZAF Base Guild – forms the basis for the new Game Guild objects.
3. ZAFAction - These are a new class of objects within Terazona. Actions create Game State Messages that are not associated with specific Entities. They are lightweight and transient, and not persisted to the GameDB or the AuditDB.

Zona Modeler Inputs & Outputs

Zona Modeler takes in three inputs. These are:

1. Zona Model Object Definition Schema – This is the XML-based model file created using the ZMUI. This file contains element and attribute definitions as well as some external information. This input object is referred to as D.

   In the demo TrackerClient, for example, this object is stored in the file:
   `%ZONA_HOME%\samples\TrackerClient\TrackerSchema.xml` (Windows)
   `$ZONA_HOME/samples/TrackerClient/TrackerSchema.xml` (Solaris/Linux)

2. Zona Model Configuration – This is the XML-based environment configuration file for the Zona Modeler network data objects. This contains information such as the working directory and the database location for Zona Modeler objects (ZonaDB). Note that although they can be located within the same database server, the ZonaDB and the Game DB can be hosted on different machines. This input is referred to as C.

   Although it's possible to specify multiple database configurations for different Zona Modeler projects, the default Terazona installation uses a single file:
   `%ZONA_HOME%\config\ZonaModelerConfig.xml` (Windows)
   `$ZONA_HOME/config/ZonaModelerConfig.xml` (Solaris/Linux)

3. Audit Model Configuration – This is the XML-based environment configuration file for the audited Zona Modeler network data objects. This contains information such as the working directory and the database location for audited Zona Modeler objects (AuditDB). Note that although they can be located within the same database server, the ZonaDB, the Game DB, and the AuditDB can be hosted on different machines. This input is referred to as C_A.

During its compile phase, Zona Modeler combines all three inputs to derive specific object representations suitable for deployment across Terazona. Specifically, Zona Modeler outputs C++ and Java code definitions for three Data Objects:

1. D_S – This is the server-side view of the D data object. This will encapsulate the System, Public, and Private Properties of specific data object instances.
2. D_C – This is the client-side view of the D data object. This will encapsulate the Public, and Private Properties of specific data object instances.
3. D_A – This is the audit view of the D data object. This will encapsulate those System, Public, and Private Properties of specific data object instances designated for auditing within C_A.

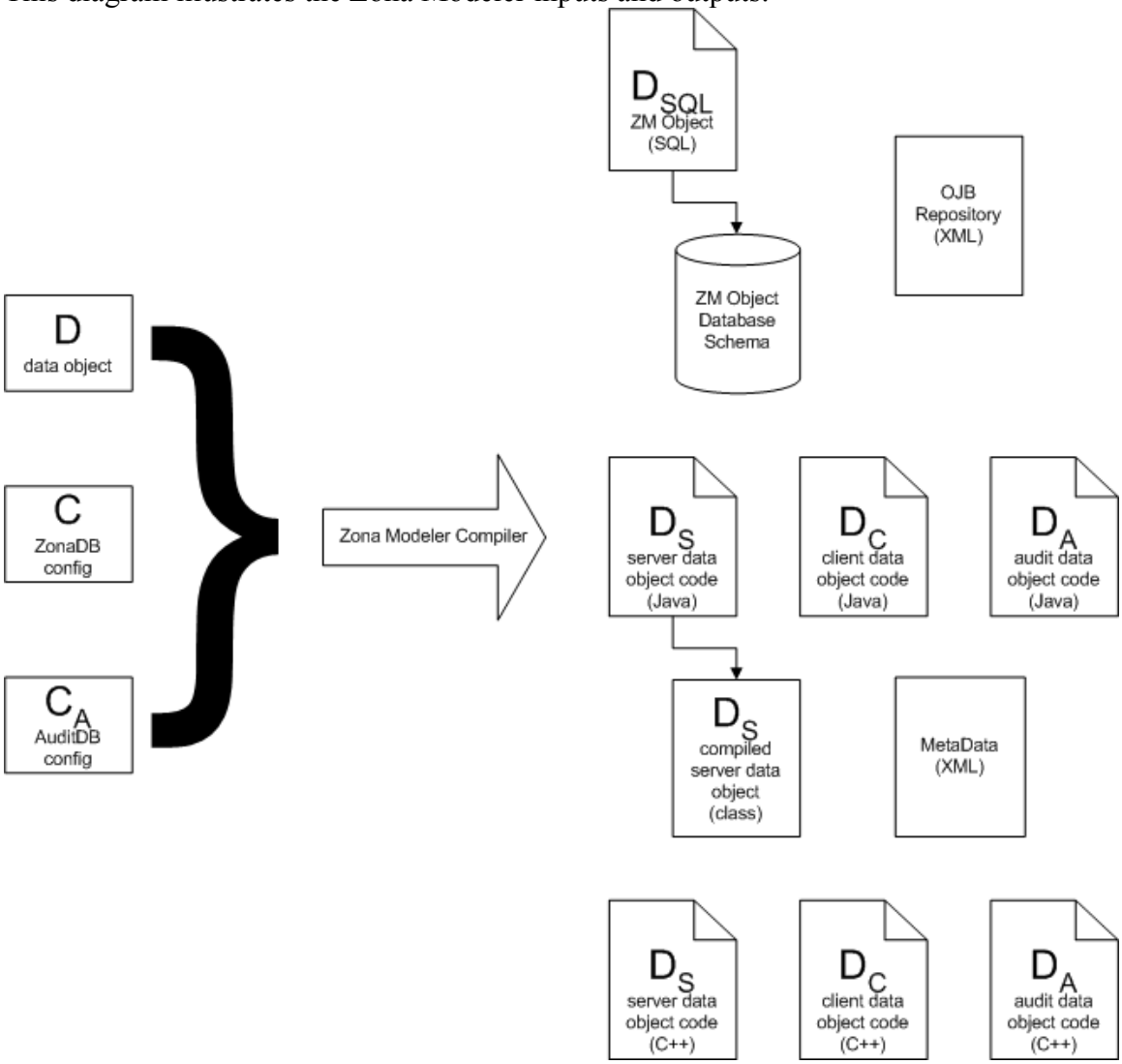
Zona Modeler also produces associated support and definition files. These are:

1. A compiled version of D_S, for deployment across all Game State Servers (GSSs).
2. A SQL definition required for Zona data object persistence. This is called D_{SQL}.
3. A database-specific schema version of XX. You can chose the database within the Zona Modeler UI.

4. An XML-based MetaData file that contains the static data necessary to instantiate objects, for deployment across all GSSs.
5. An XML-based ObJectRelationalBridge (OJB) file that describes the object-relational persistence mappings that link Zona Modeler objects with specific database columns.

Zona Modeler does not compile the C++ versions of D_S, D_C, or D_A. If necessary, you can use your C++ compiler of choice here.

This diagram illustrates the Zona Modeler inputs and outputs:



Zona Modeler Run Time

During run-time, Zona Modeler calculates how to most efficiently encode state changes for a game object. It then marshals the network game objects across the wire "by value", as bitstream encodings of object state change deltas. Accompanying each bitstream it sends a short int value that encodes a Class Id.

During run-time unmarshalling, the receiving machine uses Zona Modeler to extract the short int Class Id value. Using this Class Id, it queries its deployed metadata and persistence mapping information to extract or create an updated instance of that object. The new game object state has thus been successfully sent across the network.

Zona Modeler Initialization

Following successful installation of Terazona, to begin developing with Zona Modeler you must complete some initialization steps:

1. First you must create a suitable database repository to persist the Zona Modeler objects. Execute this script:

   `%ZONA_HOME%\sql\mssql\createZonaSchema.bat` *`host sa pass`* (Windows)
   `$ZONA_HOME/sql/mssql/createZonaSchema` *`host sa pass`* (Solaris/Linux)

host – the hostname with the target database
sa – the database user with authority to create Tables in this database
pass – the password for the authorized user

   This creates the database and makes a suitable USERDIRECTORY table.

2. Then you must initialize the demo applications. Within each sample project directory, execute the `setup.bat` (*Windows*) or `setup` (*Linux/Solaris*) script. This will create version-specific metadata and create any required user accounts by calling `createDemoUser`. Modify the parameters to suit your development requirements.

3. Next you must build the sample applications. Within each sample project directory, execute the `MakeRelease.bat` (*Windows*) or `MakeRelease` (*Linux/Solaris*) script. For debugging, you can use the `MakeDebug.bat` (*Windows*) or `MakeDebug` (*Linux/Solaris*) scripts.

4. Load the Zona Modeler UI. On Windows you can use the Start Menu icon. The line command to load the Zona Modeler UI is:

   `%ZONA_HOME\bin\startZonaModeler.bat` (Windows)
   `$ZONA_HOME/bin/startZonaModeler` (Solaris/Linux)

Zona Modeler Design Time

Zona Modeler design time consists almost exclusively of interacting with the Zona Modeler UI (ZMUI). This is a combined editor and Zona Modeler object compiler. During Zona Modeler Design Time, game designers or developers use ZMUI to create or modify Game XML file that describes all of the network game objects. Additionally, ZMUI creates or modifies associated Game Configuration XML files that describe how the network game objects are persisted, both for the Zona Modeler Database (ZMDB) and the Audit Database (AuditDB).

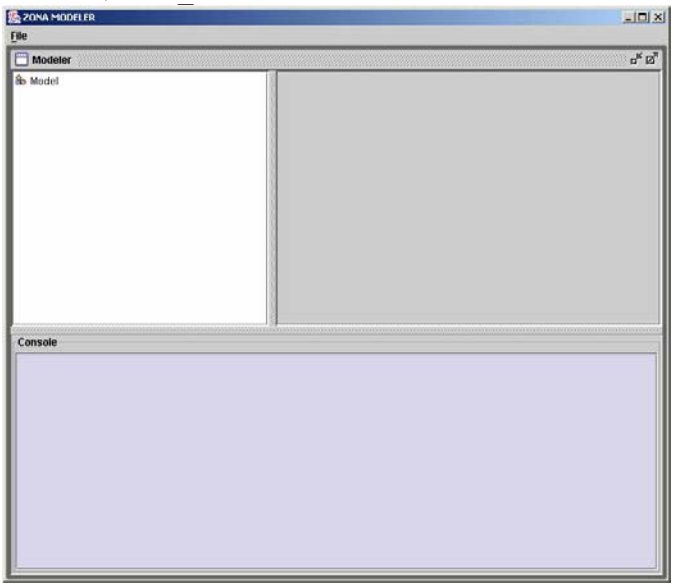
Zona Modeler User Interface

All the design-time operations of Zona Modeler are automated through the Zona Modeler User Interface (ZMUI). The ZMUI allows enables non-programmatic creation and management of network game objects. Game designers can model game objects while game developers work on game logic and designers produce sound and graphic assets.

On Windows within the Terazona Servers Start Menu there is a shortcut icon to start ZMUI. You can also start ZMUI using this script:

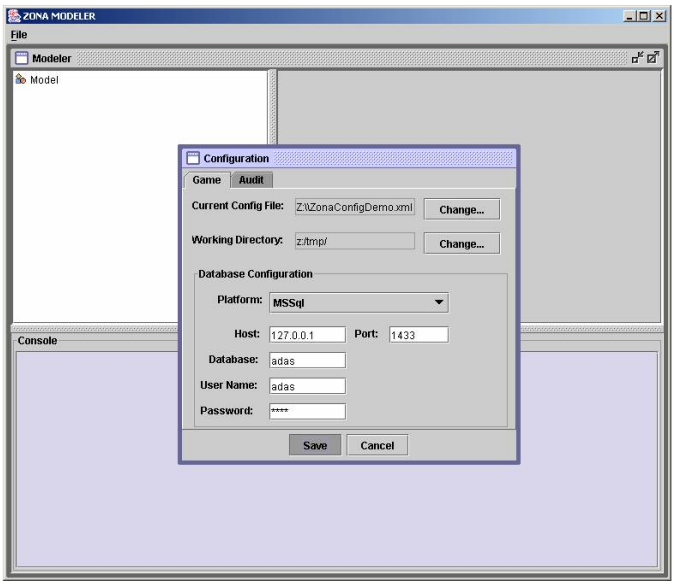
`%ZONA_HOME%\ZonaHome\bin\startZonaModeler.bat` (Windows)

`$ZONA_HOME/ZonaHome/bin/startZonaModeler` (Solaris/Linux)

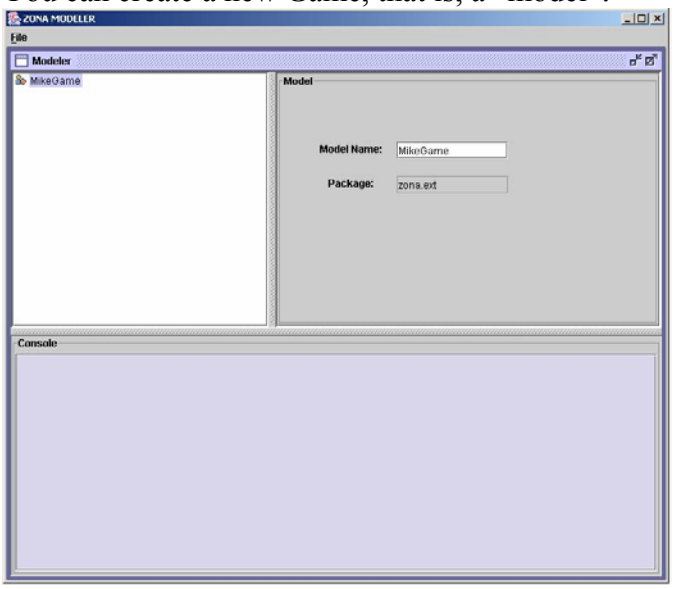


Within ZMUI, you create the Game Object Model file that defines the elements and attributes of the network objects used by Terazona during your game.

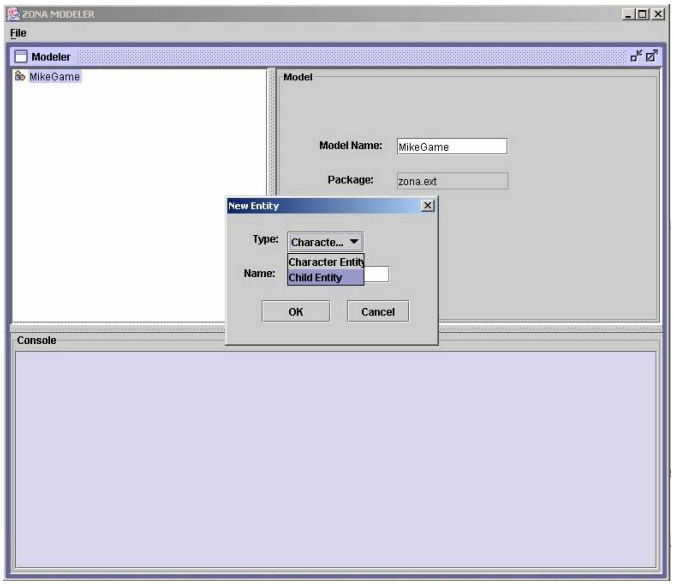
You can edit the Zona Modeler configuration information:



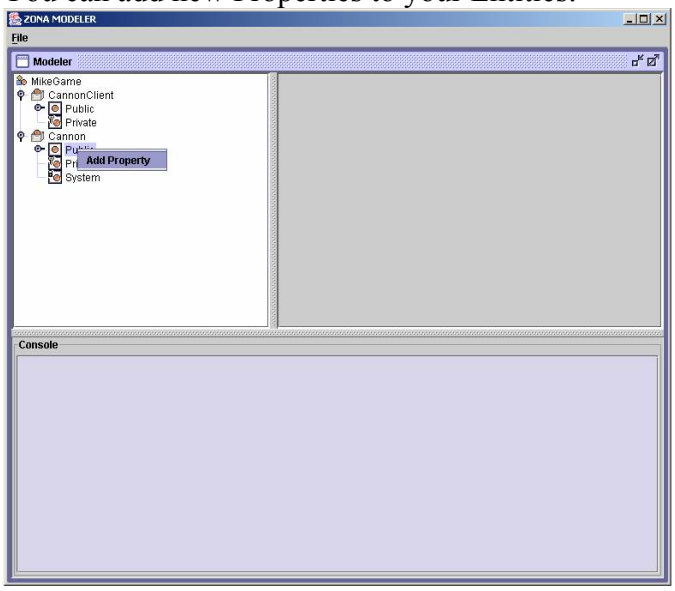
You can create a new Game, that is, a “model”:



You can create new Entities:

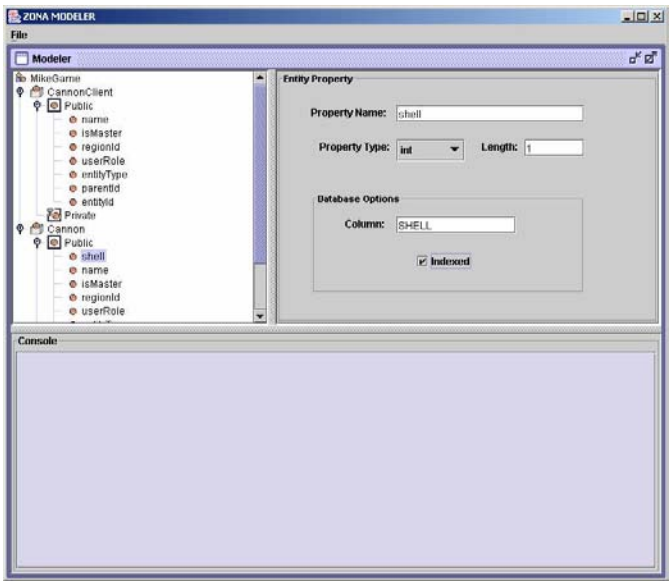


You can add new Properties to your Entities:



Note that within ZMUI, the Child (D_C) and Audit Entities (D_A) do not show in the ZMUI, because these are purely derived from the basic ZM data object (D) and are not user-editable.

And you can edit these new Properties:



Zona Modeler Object Deployment

Following successful execution of the ZonaModeler compile process, the following directory contains all of the ZM output files, including D_S, D_C, and D_A class files and metadata information required for runtime:

`%ZONA_HOME%\ext` (Windows)
`$ZONA_HOME/ext` (Solaris/Linux)

The ZM output is stored in subdirectories as follows:

\ext Subdirectories	**Contents**
classes	Run-time server-side Java classes
metadata	Run-time object instantiation information
ojb	Run-time object-relational mapping information
src	Object source code (C++, Java). Relational table code (SQL)
debug	First-time ZM model initialization data stored as compressed archives. Useful for debugging and support.

Server Deployment

1. Deploy the contents of the **`\ext`** directory onto all servers within the Terazona cluster as follows:
   `%ZONA_HOME%\ext` (*Windows*)
   `$ZONA_HOME/ext` (*Solaris/Linux*)
2. Ensure that the **`\ext`** directory is referenced within each server's Java `CLASSPATH`.
3. Build your Server plugin incorporating the generated C++ code (.h and .cpp) from:

a. `%ZONA_HOME%\ext\src\cpp\server` (*Windows*)
   `$ZONA_HOME/ext/src/cpp/server` (*Solaris/Linux*)
b. `%ZONA_HOME%\ext\src\cpp\common` (*Windows*)
   `$ZONA_HOME/ext/src/cpp/common` (*Solaris/Linux*)

Client Deployment

1) Build your Client executable incorporating the generated C++ code (.h and .cpp) from:

a. `%ZONA_HOME%\ext\src\cpp\client` (*Windows*)
   `$ZONA_HOME/ext/src/cpp/client` (*Solaris/Linux*)
b. `%ZONA_HOME%\ext\src\cpp\common` (*Windows*)
   `$ZONA_HOME/ext/src/cpp/common` (*Solaris/Linux*)

During compile-time, the metadata and object-relational data is stored within your executable. As a result, these files do not require separate client-side deployment.

Game Guilds

Game Guilds enable game developers to group Entity state updates across the Terazona cluster in terms of sets of characters or entities. Previously in Terazona, Entity state updates were spatially based, in that Entities could update properties for those Master or Ghost Entities that were in Regions defined as neighboring. Game Guilds avoid this spatial limitation by allowing flexible Entity membership irrespective of in-game Region geography or Entity location.

In terms of management and membership, Game Guilds are similar to the extant Chat Guilds. The key difference is that while Characters and Entities within Chat Guilds can exchange only text information, Characters and Entities within Game Guilds can exchange all Entity state data with other members of the Game Guilds.

For example, Game Guilds enable game designers to specify an in-game "religion" or "faction" that is aligned to a specific deity. In return for pledging allegiance to this deity, member Characters can receive periodic or permanent alterations to their Character Properties, such as an increase in "manna". "luck", "karma", or any game-specific Character attributes. They can also be used to cast a particular "spell" on all members of that Game Guild.

Game Guilds can also be used for player-to-player messaging, but at this time Zona recommends continuing to use the extant Chat Guild framework for this purpose. Game Guilds are at an early stage of development and do not yet provide as much rigor and extensibility for large-scale messaging as the Chat Guilds. Developers are strongly discouraged from using Game Guild Chat functionality as a replacement for pure Chat Guilds.

Game Guild Structure

This section describes the structure and composition of the Game Guild object. Conceptually, Game Guilds object design borrows from the earlier Guild object design.

- Game Guilds can be transient or persistent.
- Additionally Game Guilds also have limited properties analogous to Chat Guilds. (that is, possessing Inviter Attributes and Guild Properties).
    1. `PERSISTENT_MEMBERSHIP` - The guild is persistent. That is, it is stored in the Game Database ("ZonaDB").
    2. `MEMBERS_CAN_POST_MSG` - Members are allowed to post messages to the Game Guild.
    3. `INV_MODERATOR` - Moderator can invite other entities to join the Game Guild.
    4. `INV_MEMBER` - Any guild member can invite other entities to join the Game Guild.
- Unlike Chat Guilds, Game Guild messages are not persisted to the Game Database.
- There can be multiple moderators for a Game Guild.

Game Guild Operations

This section defines all the operations on Game Guilds. Operations include Guild Management and Guild Messaging

Guild Management

The Game Guild management enable developers to provide a management framework for players' Characters to create and manage Game Guilds and game Guild memberships.

- Create Guild – Client can create a new Guild
- Delete Guild - Clients with proper permission can delete a guild.
- Join Guild – Clients with proper permission can join a specific guild.
- Leave Guild – Clients can leave a guild.
- Invite to Guild – Sends out specific invitations to invite a player to join a guild.
- Remove from Guild – Can be used to kick out players from a guild.
- Add / Remove Guild Moderator – Adds and removes moderators. A Game Guild can have multiple moderators.
- Fetch Guilds –Asynchronously fetches the clients guilds by firing add guild callbacks on the client. Because of the lack of the SELF_INVITER attribute in Game Guilds, this function returns only those Game Guilds of which the calling Character is already a member.

Guild Messaging

The primary purpose of Game Guilds is to send Game State Updates to all subscribed, member Entities. You should use Chat Guilds to send plain text communications between Entities.

- Send Guild Game State – Sends a Game State Message to a particular Game Guild. These messages are not be distributed in the traditional Region-specific distribution, but instead are broadcast only to the members of the Game Guild.
- Enter Guild –Activate a client within a Game Guild.
- Exit Guild – Deactivate a client within a Game Guild. As a consequence of player logoff, that player's Character exits all Game Guilds.
- Monitor Guild – Sets a callback handler for Game Guild events.
- Game Guild Callbacks
    - On Member Enter Guild.
    - On Member Exit Guild.
    - On Guild Game State Message.
    - On Add Guild.
    - On Remove Guild.
    - On Add Moderator.
    - On Remove Moderator.
    - On Guild Invite.
    - On Guild Chat Message.
    - On Game Guild Error Message.
- Guild Error messages – Errors in guild operation are reported asynchronously in the form of error callbacks. The errors contain this data:I

- Information about the Game Guild operation.
- The Game Guild Id.
- The Id of the Entity that attempted to perform the operation.

Game Guilds API Outline:

Refer to following header files for Game Guild functionality:

Game Guild Services – `GameGuildServiceInterface.h`
Game Guild Callbacks – `GameGuildCallback.h`

The ZonaClient… header files declare the Client classes extending from corresponding ZonaBase… classes. The client classes are empty implementations - all the functions are defined in the Base class implementation. The ZonaGuildChatMsg class is used by Clients to encapsulate chat messages.

Entity Behavior In Relation To Game Guilds

- Character Entities and Child Entities can belong to multiple Game Guilds, and can be active in (that is, subscribed to updates from) all Game Guilds at any time.
- The Game Guild memberships of entities are retained during Child transfers.
- During Entity deletion, the Entity's membership of all Game Guilds is cancelled and the Entity is removed from all of its Game Guilds.
- When Guild members enter or exit the Game Guilds, then the other Game Guild members receive a CAPI notification. The entering member will receive the same notification for all other logged-in Game Guild members.
- Entity updates to a sphere are not automatically reflected in all the Game Guilds that the Entity has membership.
- Forum Chat Guilds, private messages, and persistent messages should still be provided using the Chat Server. The Chat services API is completely different from the Game Guild Services and is subclassed completely differently, despite some similar function names.
- The functionality and performance of Game Guilds will improve in forthcoming releases of Terazona 1.2.x.
- Game Guild management has been incorporated into the GSS. The functionality can be accessed by a new set of Game Guild APIs that have been built into CAPI. Typically, the players create Game Guilds at runtime.

Client API Changes

ZonaServices.h Additions

Function	Reason
int publishEntityPropertyUpdates (int entityId)	Publish updated Properties to the Server for a specific Entity.
int publishAllEntityProperties ()	Publish updated Properties to the Server for all locally managed Entities, that is, all Entities within that Client's Zona Entity Manager (ZEM) cache.

ZonaServices.h Changes

Old	New
IntVector& getModerators (int guildId)	bool GetModerators (int guildId, IntVector& moderators)
const wchar_t* getLoginName () const	char* getLoginName() const
int monitorGameState()	int monitorGameState (gamestatecallback*)
void setGameStateCallback (GameStateCallback*callback)	int monitorEntityUpdates (entitycallback*)

GameStateManager.h Deletions

The entire GameStateManager queue management class has been removed because the new CAPI autoupdate functionality replaces and supersedes the old GameStateManager functions.

Function	Reason
GameStateManager (int maxSize= GAMESTATE_MANAGER_MAX_MSG)	Replaced and superseded.
virtual ~GameStateManager ()	Replaced and superseded.
void pushGameState (int entityId, char* data, short dataSize)	Replaced and superseded.
bool popGameState (int* entityId, char*& data, short* dataSize)	Replaced and superseded.
int queSize ()	Replaced and superseded.

GameStateCallback.h Changes

Function	Reason
virtual void onPlayerReset (int resetAction, int resetReason)	Moved to Callback::onNotifyPlayerReset().

ZonaClientEntity.h Additions

Function	Reason
int publish()	Because Zona Modeler now tracks altered property data, when you request a publish (), the Terazona Client sends all Entity property updates to the server. This is a convenience method that actually calls ZonaServices:: publishEntityPropertyUpdates(entityId).

ZonaClientEntity.h Deletions

Function	Reason
void setPublicDisplayPropertyDirty ()	Zona Modeler now tracks altered property data. You do not need to explicitly set “dirty” flags. You request the Client publishes updated property data to the Server Plugin by calling ZonaClientEntity::publish().
void setPublicStatePropertyDirty ()	Zona Modeler now tracks altered property data. You do not need to explicitly set “dirty” flags. You request the Client publishes updated property data to the Server Plugin by calling ZonaClientEntity::publish().
void setPrivatePropertyDirty ()	Zona Modeler now tracks altered property data. You do not need to explicitly set “dirty” flags. You request the Client publishes updated property data to the Server Plugin by calling ZonaClientEntity::publish().
copyFromZO (ZonaObject*zo)	This was for internal use and is no longer required.
copyToZonaObject (ZonaObject*zo)	This was for internal use and is no longer required.

ZonaClientCharacter.h Deletions

Function	Reason
void copyFromGC (GameCharacter*gc)	This is no longer required.

ZonaClientEntity.h Changes

Old	New
int publish()	Because Zona Modeler now tracks altered property data, when you request a publish (), the Terazona Client sends all Entity property updates to the server. This is a convenience method that actually calls ZonaServices:: publishEntityPropertyUpdates(entityId).

Game Server API Changes

ZonaGSValidate.h Changes

Old	New
void onValidateEntityPropertyUpdate (ZonaEntity* entity, int propertyFlag, byte* property, short size)	bool onValidateEntityPropertyUpdate (ZonaServerEntity* entity, ZonaServerEntity* previousEntity)

Game Server API Additions

GameGuildServiceInterface.h Additions

Function	Purpose
GameGuildService (ZonaClient* aClient)	Constructor
virtual ~GameGuildService()	Destructor
void createGuild (ZonaClientGuild* guild, int entityId)	Creates a new Guild in the game
void deleteGuild (ZonaClientGuild* guild, int entityId)	Deletes a specified Game Guild
void requestMemberGuilds (int entityId)	Asynchronous request to fetch Game Guild memberships for a specified Entity
void joinGuild (int guildId, int entityId)	Request to join the specified Game Guild
void leaveGuild (int guildId, int entityId)	Request to leave the specified Game Guild
void enterGuild (int guildId, int entityId)	Request to enter the specified Game Guild
void exitGuild (int guildId, int entityId)	Request to exit the specified Game Guild
void inviteMember (int guildId, int inviteeId, int inviterId)	Request to invite specified member to specified Game Guild
void removeMember (int guildId, int targetEntityId, int memberId)	Request to remove existing, specified member from specified Game Guild.
void monitorGuildMessage (GameGuildCallback* guildCallback)	Set a callback for listening to Game Guild messages
void stopMonitorGuildMessage()	Stop listening to Game Guild messages
void addGuildModerator (int guildId, int entityId, int moderatorEntityId)	Request to add a moderator to a Game Guild
void removeGuildModerator (int guildId, int entityId, int moderatorEntityId)	Request to remove a moderator from a Game Guild
void sendChatMessage (ZonaGuildChatMsg* aMsg)	Send a chat message to the Game Guild members
void sendGuildGameState (int guildId, int entityId, char* state, int stateLen)	Send a Game State message to the Game Guild members
ZonaClientGuild* getGameGuild (int guildId)	Fetch an instance of a Game Guild

GameGuildCallback.h Additions

Function	Purpose
GameGuildCallback()	Constructor
~GameGuildCallback()	Destructor
virtual void onAddGuild (ZonaClientGuild* guild)	Called by ZonaServices in response to a request to fetch member Game Guilds, or if an invite has been sent by another member
virtual void onRemoveGuild (ZonaClientGuild* guild)	Called by ZonaServices when any of following actions occur: The Game Guild is removed by a moderator This member is removed by moderator The member exits or leaves the Game Guild
virtual void onMemberEnterGuild (ZonaClientGuildMembership* aShip)	Called by ZonaServices when a member enters a Game Guild
virtual void onMemberExitGuild (ZonaClientGuildMembership* aShip)	Called by ZonaServices when member exits a Game Guild
virtual void onAddModerator (int guildId, int moderatorId)	Called by ZonaServices when a member is set as a moderator
virtual void onRemoveModerator (int guildId, int moderatorId)	Called by ZonaServices when a member is removed as a moderator
virtual void onGuildInvite (int guildId, int inviterId, int inviteeId)	Called by ZonaServices when a member sends a Game Guild invite to this member
virtual void onGuildGameState (int guildId, char* state, int stateLength, int entityId)	Called by ZonaServices when a member sends a Game Guild Game State Message
virtual void onGuildChatMessage (ZonaGuildChatMsg* msg)	Called by ZonaServices when a member posts a Game Guild Chat Message
virtual void onGuildErrorMessage (int errorId, int guildId, int operationId, int entityId)	Called by ZonaServices when a user action results in an error condition

ZonaBaseGuild.h Additions

Function	Purpose
ZonaClientGuild	Constructor
virtual ~ZonaClientGuild()	Destructor
virtual int getPropertyAttributes()	Get the Game Guild's Property attributes
virtual int getInviterAttributes()	Get the Game Guild's Inviter attributes
virtual int getGuildId()	Get the Game Guild's Attributes
virtual char* getGuildName	Get the Game Guild's Name
virtual void setPropertyAttributes (int attr)	Set the Game Guild's Property attributes
virtual void setInviterAttributes (int att)	Set the Game Guild's Inviter attributes
virtual void setGuildName (char* name)	Set the Game Guild's Name
virtual void setGuildId (int id)	Set the Game Guild's Id
ZonaBaseGuildMembershipMap* getMemberMap()	Fetch the Game Guild Membership Map
ZonaBaseGuildMembership* getMember (int memberId)	Fetch the Game Guild membership of a specified member
bool isMember(int entityId)	Utility function to check if an Entity is member of this Game Guild
bool isPersistable()	Utility function. Checks for Persistable Game Guild Property attribute
bool canMembersPostMessages()	Utility function. Checks if members can post messages to the Game Guild.
bool isMemberInvite()	Utility function. Checks if members can invite other members to join the Game Guild.
bool isModeratorInvite()	Utility function. Checks if moderators can invite other members to join the Game Guild.
static ZonaBaseGuild createClientClassFromId (short classId)*	Internal ZONA use. Required by CAPI. May be deprecated. PLEASE DO NOT USE.
ZonaBaseGuildMembership addMember (ZonaBaseGuildMembership* mShip)*	Internal ZONA use. Required by CAPI. May be deprecated. PLEASE DO NOT USE.
void removeMember (int memberId)	Internal ZONA use. Required by CAPI. May be deprecated. PLEASE DO NOT USE.

ZonaBaseGuild.h Property Attributes

Attribute	Purpose
PERSISTENT_MEMBERSHIP	Game Guild and Memberships should be persisted to GameDB. If not set, then the Game Guild will be treated as transient
MEMBERS_CAN_POST_MSG	Members can actively send messages to the Game Guild

ZonaBaseGuild.h Inviter Attributes

Attribute	Purpose
INV_MODERATOR	Moderators can invite members to join the Game Guild
INV_MEMBER	Members can invite other members to join the Game Guild

ZonaClientGuildMembership.h Additions

Function	Purpose
ZonaClientGuildMembership ()	Constructor
virtual ~ZonaClientGuildMembership ()	Destructor

ZonaBaseGuildMembership.h Additions

Function	Purpose
ZonaBaseGuildMembership ()	Constructor
virtual ~ZonaBaseGuildMembership ()	Destructor
virtual int getMemberId ()	Fetch this member's EntityId
virtual int getGuildId ()	Fetch this member's Game Guild Id
virtual bool isModerator ()	Check if member is a moderator
virtual short getMemberClassId ()	Fetch this member's Class Id
virtual short getGuildClassId ()	Fetch this member's Game Guild's Class Id
virtual void setMemberId (int value)	Set the member's Entity Id in the GuildMembership object that encapsulates the association between a Game Guild and an Entity
virtual void setGuildId (int value)	Set this member's Game Guild Id
virtual void setMemberClassId (short value)	Set this member's Class Id (that is, the ZonaModeler class identifier for the Member's Entity Class Id
virtual void setGuildClassId (short value)	Set this member's Game Guild's Class Id (that is, the ZonaModeler class identifier for the associated Game Guild)
virtual void setIsModerator (bool value)	Set this member's moderator status
static ZonaBaseGuildMembership createNewClientInstance()*	Internal ZONA use. Required by CAPI. May be deprecated. PLEASE DO NOT USE.

ZonaGuildChatMsg.h Additions

Function	Purpose
ZonaGuildChatMsg (int guildId, int senderId, char* subject, int subLen, char* body, int bodyLen)	Constructor
virtual ~ZonaGuildChatMsg ()	Destructor
int getGuildId ()	Get the Game Guild Id from the message
int getSenderEntityId ()	Get the sender's Entity Id from the message
int getSubjectLength ()	Get the message's Subject data length
int getBodyLength ()	Get the message's Body data length
char* getSubject ()	Get a pointer to the message's Subject data
char* getBody ()	Get a pointer to the message's Body data