

Use this guide to understand the expectations of the millions of gamers who will be downloading and playing your game, and the actions you can take to make your game look, feel, and play its very best.

Note: This guide references features in XNA Game Studio 3.1. A guide for XNA Game Studio 4.0 is being developed, but isn't available yet.

A few things to remember:

- The sections in this guide are ordered by priority, but everything in it is important. Read it all and verify that your game takes these issues into account.
- These best practices are suggestions and not requirements, but in many cases your game will be unplayable if you don't follow them. They aren't necessary to pass peer review and reviewers shouldn't use these guidelines to pass or reject games, but reviewers may reject games that are unplayable.
- Use the playtest support available in the submission process on the App Hub to have other developers validate and catch issues with your games, before millions of gamers do.

Gamers Expect Games to "Just Work" on Any TV

Background: Xbox 360 consoles can be plugged into TV sets of all types, with many different resolutions, aspect ratios, and more. Overscan (edges of the TV screen that don't draw the entire viewport), stretching of your game's visuals, and even crashes, can occur if resolutions aren't handled properly.

Actions:

- Set your game's resolution to 1280 x 720 (720p native resolution) to work on all TVs. This gives you a single resolution target for all your art content.
- Draw the entire scene to your Viewport size, but draw your critical gameplay features (HUD, main character, and so on) to the region inside
- **Viewport.TitleSafeArea.**
- If using text, use a 14-point or higher font to ensure the font can be read on standard-definition TVs. Draw the **SpriteFonts** at full size.

For reference, see:

- [Xbox 360 Programming Considerations](#)
- [Safe Area](#)

- [Displays, Client Bounds, Viewports, and Back Buffers](#)

Gamers Expect a Good Trial Mode

Background: Many gamers want to try before they buy, and the Xbox LIVE Indie Games system allows players a timed trial of your game before they buy. Currently, the trial mode time is set at eight minutes, although this may be changed in the future. When the time runs out, the player is shown a screen where he or she can either purchase the full game, or exit. The only additional restriction in Trial Mode is that Xbox LIVE matchmaking is disabled. During the trial, you have the opportunity to show off the very best your game has to offer.

Actions:

- Check whether the game is in Trial Mode by checking if the **Guide.IsTrialMode** property is **true**. This will always start out as true, and can change shortly after game startup, or as part of a new player signing in, or in response to an in-game purchase. Be sure to check this once every frame, as opposed to just once at startup.
- Consider ways to give the player the best experience in the limited amount of time available. This may mean avoiding title screens or exposition on the story and getting right to the gameplay as soon as possible.
- When the purchase screen opens, Xbox LIVE Indie Games will set **Game.IsActive** to **false**. If the player chooses to buy the game and a successful purchase is made, the game will resume and **Guide.IsTrialMode** will be set to **false**. If you wish, you can manually activate the purchase screen before the time limit expires by calling **Guide.ShowMarketplace**. Your game must be able to handle the **IsTrialMode** changing without re-starting. This may mean that you need to reload or re-initialize the menus, gameplay, or other aspects of the game when the flag changes, without disrupting the game.
- Consider displaying a scene as the gamer exits the game that gives the gamer the option of being taken to Marketplace to purchase the game by using the **Guide.ShowMarketplace** API, if the **Guide.IsTrialMode** property is **true**.
- If your game has Xbox LIVE multiplayer support, disable it at the menu if the game is in Trial Mode; if the player selects the option, you can

call **Guide.ShowMarketplace**. Remember to re-enable it if the **Guide.IsTrialMode** flag changes to **false** during the game.

- Test out trial mode by setting **Guide.SimulateTrialMode** to **true**, or by using the “Play Trial Game” option to launch your game from the Xbox Dashboard. This forces the **Guide.IsTrialMode** flag to **true**. You can also test that the game handles the transition from Trial mode to being purchased by changing the **Guide.SimulateTrialMode** to **false** during game play by using a button on the controller that game doesn’t ordinarily use. This way, you know that the game will work properly when they purchase the game without re-starting.

For reference, see:

- [Network Game State Management](#)

Gamers Expect a Friendly Menu and Control System

Background: Games are essentially a cycle. From the Main Menu (and alternate menus such as Options and Help), gamers expect to enter a game session, play, and then return to the main menu when they are done. Any deviation from this should be intuitive and easy to remember.

For menu navigation, gamers expect to use both the stick and the D-Pad, with **A** being the “accept” or “go forward” button, and **B** and **Back** being the “cancel” or “go back” buttons.

Both successful and unsuccessful menu operations should be reported in some way to the user, in both graphical and audio form. If there are errors, they should be provided to the user in simple, prescriptive language that tells the player what to do.

Actions:

- Use a Menu system, such as in the [Game State Management Sample](#).
- Use the stick and D-pad to move the menu selector, and use **A** and **B/Back** to choose menu items or cancel, respectively. When using the stick for menu movement, poll with a delay to keep the menu selector from moving too fast.
- The active menu item should stand out clearly from other menu options. Use patterns, animations, or other graphical items to support color blind players.

- Make sure confirmations and errors are clearly shown to the player, with guidance about what the player should do next.
- Consider changing the **Back** button on the controller, which by default exits the game with no warning. Provide either a confirmation dialog to insure that the player really wanted to leave the game, or provide an exit menu option from the main menu and remove the **Back** button exit functionality entirely.
- The game should bring up a pause menu if the active controller becomes disconnected, or if the player presses the Start button.
- In networked games, gameplay should continue behind the pause menu even though the local player is no longer engaged in the action.
- Make sure the game respects **Game.IsActive** and pauses as appropriate when this is set to **false**, such as when the guide is being displayed over the game.
- Games should not stop or pause without game-representative, on-screen graphics. When the pause time is greater than five seconds, the game must indicate the reason and provide on-screen feedback/explanation (Loading Screen, for example).

For reference, see:

- [Game State Management Sample](#)
- [How To: Pause a Game](#)
- [How To: Detect Whether a Controller Is Disconnected](#)

Gamers Use One Xbox 360 Controller

Background: This may seem obvious, but it is important. Not only do gamers expect to use an Xbox 360 Controller (not a plugged-in keyboard or Chatpad), but in a single-player game, they are going to expect whichever controller they pick up to be the one the game recognizes throughout their gaming session.

Actions:

- Be aware that the player may not be using the controller that is assigned to **PlayerIndex.One**. To detect which controller the player is using, create a splash screen or initial menu that prompts to the player to press A or Start. Detect which PlayerIndex pressed the button to start the game and use that as the active controller. Here is an example of code to do this:

```

PlayerIndex controllingPlayer = PlayerIndex.One;

for (PlayerIndex index = PlayerIndex.One; index
<= PlayerIndex.Four; index++)
{
    if (GamePad.GetState(index).Buttons.Start
== ButtonState.Pressed)
    {
        controllingPlayer = index;
        break;
    }
}

```

- If your game requires a signed-in player profile, you can then look up the profile details for the selected `PlayerIndex`, or if no profile is currently signed in at that index, call **Guide.ShowSignIn** to let the player select their profile:

```

SignedInGamer gamer = Gamer.SignedInGamers[controllingPlayer];

if (gamer != null)
{
    playerName = gamer.Gamertag;
}
else
{
    Guide.ShowSignIn(1, false);
}

```

- Don't require the keyboard for input.
- Games should launch successfully and get to their menu and/or attract mode even if a controller isn't active (for example, if the game is launched with an IR remote).
- Use **Guide.ShowSignIn** to support the signing-in of multiple players to a local multiplayer game.

For reference, see:

- [Game State Management Sample](#)
- [Networked Game State Management Sample](#)

Be Kind With Audio

Background: Audio, at first glance, looks like a very easy technology to get right. Drop in sounds, and play them when asked. However, audio systems are as diverse as TVs. Volume, position, and frequency ranges may be different across the different systems.

Don't play sounds with wildly different volumes, forcing players to turn the TV volume up or down.

In addition, some gamers don't like music in their games and will want to turn it off or replace it with their own music through the Xbox 360 Guide.

Actions:

- Test your game audio on as many audio configurations as possible (stereo, mono, 5.1, and headphones).
- Identify and eliminate major volume changes between sounds that would force the player to change audio volume while playing your game. Do this by leveling your sounds to a standard sound. The Xbox 360 startup sound is a good reference. Adjust your speaker volume so that the Xbox 360 startup sound is loud, but not overly so, and adjust your game sounds to match that level.
- If you have music in your game, tag it as such. If you're using XACT, drag the music sounds into the Music category in the tree. If you use the Sound Effects API, load your background music files as Song objects, and play them with **MediaPlayer.Play**. These methods ensure that the player can replace the music through the Xbox 360 Guide.
- Here's a tip to lower your game's binary size if you're using XACT: enable XMA compression on your wave banks. Add compression in your XACT project in the "Compression Presets" node, and choose the XMA format. Edit the properties of your wavebanks, setting their "Compression Preset" box to the new preset.
- Be aware that **MediaPlayer.Play** on the Xbox 360 is asynchronous, which means that the song doesn't immediately start playing. Indeed, you can check the next frame and the song almost certainly won't have started playing by then, and the **MediaPlayer.State** will probably still be **MediaState.Stopped**! If you want to start a new song when the old one finishes and you check every frame for the current song to be done, each frame you'll change songs if you aren't careful. Instead, add an eventhandler to the **MediaPlayer.ActiveSongChanged** event and when

this event fires, you can begin checking for the current song to be done by using **MediaPlayer.State == MediaState.Stopped**.

- The content processor for the XNA Framework sound effect API defaults to high compression. Consider how compression affects the overall size and load times of your game, versus the quality of the audio.

Gamer Profiles Matter

Background: Gamers spend a lot of time and effort on their Xbox 360 Profiles. Even if they're not connected to Xbox LIVE, profiles are used in-game to identify which player is which, to store saved games, to identify a gamer in a high score list, and, in some cases, to store game defaults such as controller sensitivity. Use these profiles to identify with your gamer, and do things that personalize the experience.

Actions:

- The key to profile is the **GamerServicesComponent**. Add it to your **Game** component list. Then you can track profiles in your game.
- When you start your game, check **Gamer.SignedInGamers** to check your active controller for a profile. You can choose to ask the gamer to sign in if a profile isn't returned, or you can allow the gamer to play anonymously. Unfortunately, anonymous play prevents you from opening a storage container for saving game progress, or looking up the player name (for a high score chart, for example) without the player having to enter their name manually.
- Use the gamer's name. Check **SignedInGamer.Gamertag** and use it in your game, such as in a status bar or in high score charts.
- The **SignedInGamer** class holds a variety of information about your player, including a **GameDefaults** property that may be useful in automatically tuning your game's control scheme or difficulty. The **GameDefaults.InvertYAxis** property is of special interest to 3D games.
- There's even more data inside **SignedInGamer.GetProfile**, such as their gamer picture. You can use this at your discretion.
- Respond to **SignedInGamer.SignedOut** on the active controller **PlayerIndex**. If the player signs out, you can decide what to do. Most games return to the main menu.

For reference, see:

- [Network Game State Management Sample](#)
- [Gamer Services Overview](#)
- [Microsoft.Xna.Framework.GamerServices Reference](#)

Represent Your Game

Background: You worked hard to create your game, so work just as hard to present your game attractively in Xbox LIVE Marketplace. Thumbnails, Box Art, Screenshots, Videos, and Rich Presence are key components for making sure that your game shows up well on Xbox 360. Also, you'll want to know when people are playing it.

Actions:

- Have a great thumbnail. When you submit your game, you'll be able to submit a 64x64 jpg file that is your game's icon.
- There are more ways to represent your game when submitting your game to the Peer Review process. Be sure to give your very best four screenshots, a short video trailer, and box art. Take the time to create these assets. Your game's assets are the means by which the Xbox LIVE Marketplace displays your game.
- Rich Presence is a way for your game to display on Xbox LIVE while it's being played. For each **SignedInGamer** playing your game, set **Presence.PresenceMode** and, if needed, **PresenceValue**, based on what a gamer is doing in your game at any particular moment. Xbox LIVE Indie Games adds your game title automatically, so everyone knows people are having fun in your game.

Make Loading and Saving Seamless

Background: From the time the game starts to the time the gamer quits the game, there should be good interaction among players and the game. At the very least, there should be something for gamers to look at.

Most games do a great deal of loading at start-up, so be sure to show something interesting during that time. When saving games, don't pause or freeze the gameplay; keep the action moving and the disk operations in the background.

Players might use Memory Units (MUs) and expect to save their game there. You must use the Storage APIs asynchronously to avoid locking up if your users are using a Memory Unit. Be sure to test your game both with and without an MU connected.

Actions:

- Never start your game with a device selector screen. First, this is tacky and makes a poor first impression. Second, by requiring a player to hit start first, you'll know the active controller and can handle follow-up message boxes properly.
- Use an animated loading screen when loading content in your game. This keeps the player from thinking the game has crashed or locked up.
- Test that your game works when more than one memory device is present, such as having a hard drive and a memory unit, or two memory units.
- The first time a player's game data must be saved or loaded, use **Guide.BeginShowStorageDeviceSelector** asynchronously to prevent locking up.
- Don't ask for a storage device again after the first time, as this may annoy the player.
- If you're going to auto-save, notify the player of the auto-save, and tell them not to turn off their Xbox 360 Console while it's saving.
- Make sure you associate saved games to a profile, but do not associate high scores to a profile. That way, everyone can see the high scores.
- Saving and loading data in text formats can be problematic, depending on the region the Xbox 360 is set to. If you convert a floating point number to a string during a save when the Xbox 360 is set to Germany, for example, you'll get 1,234 when you might expect 1.234. To insure that numeric conversions to and from strings generate the same values in all regions, store the numbers in a culturally invariant manner, or use a binary file format instead of text.

For reference, see:

- [How To: Get a StorageDevice Asynchronously](#)
- [BeginShowStorageDeviceSelector Reference](#)
- [Network Game State Management Sample](#) (for a multithreaded loading screen demo)

Xbox LIVE Multiplayer: Make it Full Featured

Background: Multiplayer is a valuable feature in your game. It increases replay value and connects gamers all over the world. However, it can be frustrating for customers if the experience isn't optimized and made user-friendly. If you're building multiplayer into your game, be sure to go the extra mile and provide the features that make Xbox LIVE gameplay great.

Actions:

- Take full advantage of the Xbox LIVE Party system to get like-minded players together in your game. Use the **SignedInGamer.PartySize** to determine if you should inform the player that with a simple button press they can open up the Party UI and quickly send out game invites. A size of zero means you aren't in a party, one means you're all alone, and anything greater means you're ready to invite some friends. A simple call to **Guide.ShowParty()** will open the guide directly to the Party screen to streamline the invites to fellow party members.
- Support invites, as it improves game visibility and ease of joining others for gameplay. To do so, respond to the **NetworkSession.InviteAccepted** event, quitting any session that the player is in, and joining the new session by calling **NetworkSession.JoinInvited**. Players can then send each other invites (if necessary, the invited player will be given the option to purchase the game) and can join each other's games in progress.
- Don't make your player hunt for games: have a Quick Match feature. Run the matching process in the background and join up with the best session available. Make quick match the default, to improve the odds of gamers finding other players via the Xbox LIVE service.
- Have a Custom Match feature with a few parameters such as level and rule sets. Limit the number of sessions returned to make it easy for the player to pick.
- If you have a pre-game lobby, display the match settings so that players entering the lobby know what they're getting into.
- Optimize, optimize, optimize! The XNA Framework can simulate latency and packet loss; you should make sure to turn on both when testing. You should target 8Kb of bandwidth for your network game, as this will cover 90% of Xbox 360s. You should also make sure your game can continue

playing well with 500 ms of latency. For more details, see the link below to Shawn's talk.

- Hosting and/or joining a Multiplayer Game through Xbox LIVE isn't available on Xbox LIVE Silver and some Xbox LIVE child accounts, even when the game is purchased. You can determine if a particular gamer is allowed to participate in an Xbox LIVE session by checking that the **SignedInGamer** class's **Privileges.AllowOnlineSessions** flag is set to true. For example:

```
if (Gamer.SignedInGamers[activeController].Privileges.AllowOnlineSessions) ...
```

- Be sure to use the rich presence to let other gamers know that your game is being played. You can also specify where a particular player is in the game by using the rich presence strings. Rich presence will also require the **SignedInGamer'sPlayerIndex**, which is another reason why you need to poll for it at startup and save it during game play. For example:

```
Gamer.SignedInGamers[activeController].Presence.PresenceMode =  
GamerPresenceMode.AtMenu;
```

For reference, see:

- [Network Architecture: Peer to Peer](#)
- [Network Architecture: Client/Server](#)
- [Network Lobby and Chat Icons](#)
- [Network Prediction](#)
- [Network Game State Management](#)
- [Shawn Hargreaves' Gamefest 2008: Network Game Optimization](#)
- [Network Player Invites](#)

Gamers May Have Special Controllers

Background: Many types of controllers are supported by the XNA Framework. These controllers, which range from flight sticks to dance pads, all return values inside the **GamePadState** class. Gamers who have these controllers expect them to work in the types of games for which they're traditionally used.

Regardless of the special types used in your game, you should always support the default Xbox 360 Gamepad, and never require special controllers for play.

Actions:

- Determine what kind of controllers your game is best suited for; racing games should support the wheel, flying games should support flight sticks, and so on.
- When the player starts the game and you know which controller they're using, call **GamePad.GetCapabilities** and check the **GamePadCapabilities.GamePadType** returned value to understand what type of controller the player is using.
- Respond to a special controller type by changing your control scheme to best fit that type. The trigger on an Xbox 360 Controller may not be best suited to a flight stick, for instance. Use the Input Reporter (linked below) to test special controller types to understand their inputs.

For reference, see:

- [Input Reporter Utility](#)

Avatars

Background: Starting with XNA Game Studio 3.1, developers can use avatars when making their Xbox LIVE Indie Game.

Actions:

- Users may not have an avatar, so your game should handle this situation. After retrieving the **AvatarDescription** via the Avatar property on the **SignedInGamer**, the developer should check the **IsValid** property. If the description isn't valid, then the player doesn't have a valid avatar. The developer can replace the user's avatar with a random avatar by using the **AvatarDescription.CreateRandom** method, or by loading a previously created **AvatarDescription**.
- If avatars are used in your networked game, users should be able to see the real avatars of the other player. When creating a networked game that uses avatars, the developer should send each player's **AvatarDescription** across the network so that each player can display all of the other players' avatars. A player should see a player's real avatar and not a random avatar.
- Games should be tested with avatars of different sizes. A player's avatar can be a number of different heights and sizes. A developer should consider this when developing the game. Collisions may look incorrect with avatars of some sizes.
- Games shouldn't expect a player to have an avatar with a specific gender. If the game contains gender-specific text such as "she," "he," "her," and

"him," the game should determine if the player's gender is female or male by using the **AvatarDescription.BodyType** property.

- At all times, keep in mind the rules regarding avatar use.

For reference, see:

- [Avatar Use](#)