August 11, 2002 at 01:12

1. Introduction. This document describes the functions available to title developers for the purpose of online authentication to enable secure online game play. The functionality allows title developers to establish their own secure online gaming infrastructure and user account system worry-free. The functions described here are only used to achieve authentication and to provide enough infrastructure for the titles to communicate securely with other online services. The functions used to communicate with other online services, such as match-making and content download, are not discussed. here.

2. Using the Xbox Live Account and Authentication APIs. First, we must include the required header files. The online functions are contained in *xonline.h*

```
#include <xtl.h>      /* Standard Title Library functions */
#include <xonline.h>      /* Xbox Live functions */
```

3. For demonstration purposes we will now show the details of signing on. The *main* function below demonstrates the steps that a title would typically follow:

```
⟨ Global Variables 16 ⟩
⟨ Function Prototypes 58 ⟩
VOID __cdecl main( )
{
  HRESULT hr = S_OK;
  BOOL bSignedIn;
  ⟨ Initialize Devices 53 ⟩
  ⟨ Wait for Memory Units to mount 54 ⟩
  ⟨ Start up the Xbox Live APIs 4 ⟩
  ⟨ Sign in to the Xbox Live Service 6 ⟩
  if (bSignedIn) GameLoop( );
  Print(L"Signing␣off...");
  ⟨ Sign off of the Xbox Live Service 52 ⟩
  ⟨ Shut down the Xbox Live APIs 5 ⟩
  BootToDash(XLD_LAUNCH_DASHBOARD_MAIN_MENU);
}
```

4. Before using the XBox Live APIs, a title must first call *XOnlineStartup*. *XOnlineStartup* will automatically call *XNetStartup* and *WSAStartup* with reasonable defaults in order to initialize the Xbox Secure Networking Libary and Winsock respectively. If you require special parameters for those functions your title should can call them first before calling *XOnlineStartup*.

```
⟨ Start up the Xbox Live APIs 4 ⟩ ≡
  hr = XOnlineStartup(Λ);
```

This code is used in section 3.

5. When a title is through with the XBox Live APIs, it can call *XOnlineCleanup* to perform final cleanup for the online functions.

```
⟨ Shut down the Xbox Live APIs 5 ⟩ ≡
  XOnlineCleanup( );
```

This code is used in section 3.

6. Signing in to Xbox Live Service. To sign in, we call the *SignIn* function.

```
⟨ Sign in to the Xbox Live Service 6 ⟩ ≡
  bSignedIn = SignIn( );
```

This code is used in section 3.

7. Signing In. The *SignIn* function is the backbone of this example. The *SignIn* function attempts to logon the first account onto the first controller, and a guest of that account on the second controller.

```
  BOOL SignIn( )
  {
    ⟨ SignIn Local Variables 8 ⟩
    ⟨ Enumerate User Accounts 10 ⟩
    ⟨ Check if the first user account has a passcode 12 ⟩
    ⟨ Assign accounts to controllers 14 ⟩
    ⟨ Specify Services required by the title 17 ⟩
    ⟨ Initiate the authentication process 18 ⟩
    ⟨ Loop until authentication is complete 22 ⟩
    ⟨ Verify that Xbox authentication was successful 23 ⟩
    ⟨ Verify that user authentication was successful 30 ⟩
    ⟨ Verify that service authentication was successful 40 ⟩
    ⟨ Set Presence 45 ⟩      /* If we made it this far, we are in... */
    return TRUE;
  }
```

8. For error checking, we will need an HRESULT

```
⟨ SignIn Local Variables 8 ⟩ ≡
  HRESULT hr;
```

See also sections 9 and 13.

This code is used in section 7.

9. User Accounts. Each user account is represented as an **XONLINE_USER** structure. The maximum number of user accounts returned during an enumeration is `XONLINE_MAX_STORED_ONLINE_USERS`. We will store the enumerated accounts in *StoredUsers* and the number of accounts found during the enumeration in *dwNumStoredUsers*.

```
⟨ SignIn Local Variables 8 ⟩ +≡
  XONLINE_USER StoredUsers[XONLINE_MAX_STORED_ONLINE_USERS];
  DWORD dwNumStoredUsers;
```

10. The *XOnlineGetUsers* function will enumerate both the hard disk and any attached memory units looking for user accounts.

```
⟨ Enumerate User Accounts 10 ⟩ ≡
  hr = XOnlineGetUsers(StoredUsers, &dwNumStoredUsers);
  assert(SUCCEEDED(hr));
  (VOID) hr;    /* avoid compiler warning */
```

See also section 11.

This code is used in section 7.

11. Next, check if there were any user accounts. If no accounts were found, a tile must give the player the *option* of going to the online dash to create new account. In addition, it is possible for a player to actually insert/remove an MU while the title account selection UI is active. A title must call XOnlineGetUsers repeatedly to account for this. For demonstration purposes, we just boot to the account signup section of the online dash if no accounts are found

```
⟨ Enumerate User Accounts 10 ⟩ +≡
  if (dwNumStoredUsers ≡ 0) {
    Print(L"No␣user␣accounts␣found.");
    BootToDash(XLD_LAUNCH_DASHBOARD_NEW_ACCOUNT_SIGNUP);
    return FALSE;    /* Should never get here */
  }
```

12. Before signing on, the title must check accounts for passcodes. The *dwUserOptions* member of the **XONLINE_USER** structure will have the `XONLINE_USER_OPTION_REQUIRE_PASSCODE` bit set if the account has a passcode. Furthermore, the *passcode* field of the **XONLINE_USER** structure will contain the actual passcode. If an account has a passcode, the player must be prompted for it. Passcodes are for *client-side* authentication only, and the Xbox online service does not use them for authentication. For demonstration purposes, we check the first user account, make a note of any passcode, and continue.

```
⟨ Check if the first user account has a passcode 12 ⟩ ≡
  if (StoredUsers[0].dwUserOptions & XONLINE_USER_OPTION_REQUIRE_PASSCODE)
    Print(L"%S␣has␣a␣passcode", StoredUsers[0].szGamertag);
```

This code is used in section 7.

13. The Xbox Live signon API *XOnlineLogon* requires a list of exactly 4 **XONLINE_USER** accounts (1 per controller) to login in a single call. The list must be a one-to-one match of controller to player in order for the online system to recognize which player is using which controller. Any unused entries must be zeroed out. This sample shows how to authenticate a single user and a guest. First, we will need an array, *LogonUsers*, which will be assigned the two accounts. Initially, *LogonUsers* is zeroed out.

```
⟨ SignIn Local Variables 8 ⟩ +≡
  XONLINE_USER LogonUsers[XONLINE_MAX_LOGON_USERS] = {0};
```

14. Start off by assigning the first account to the first controller

⟨ Assign accounts to controllers 14 ⟩ ≡
```
  LogonUsers[0] = StoredUsers[0];
```
See also section 15.

This code is used in section 7.

15. Next, add a guest player on the second controller. A guest account is specified by first copying the sponsor account information into the controller array and then setting one of the guest bits in the *dwUserFlags* of the account `XUID`, by calling *XOnlineSetUserGuestNumber*. The second parameter to *XOnlineSetUserGuestNumber* indicates the guest number for the sponsor. It can be 1, 2, or 3, since there can be up to three guest accounts. It actually doesn't matter which of these values we use, only that no two players use the same guest number for the same sponsor. We arbitrarily chose the value *1* .

⟨ Assign accounts to controllers 14 ⟩ +≡
```
  LogonUsers[1] = StoredUsers[0];
  XOnlineSetUserGuestNumber(LogonUsers[1].xuid.dwUserFlags, 1);
```

16. Initiating the authentication process. When a title calls *XOnlineLogon* to sign in, instead of blocking until the authentication completes, an asynchronous task handle is returned. This task handle can be passed to *XOnlineContinue* to perform a unit of work when the title has some spare cycles (e.g. when waiting for the next flip or for the graphics push buffer to clear). As you'll discover later, asynchronous tasks handles are returned by many of the Xbox Live APIs. Since we will be signing on, we will need a to declare a task handle:

```
⟨ Global Variables 16 ⟩ ≡
  XONLINETASK_HANDLE g_hLogonTask;
```

This code is used in section 3.

17. As part of the authentication process a title must specify which services it will be using. You should specify the services that are appropriate for your title, but no more. Each service requires additional authentication time and network traffic. For demonstration purposes, the matchmaking service is specified. Additional services ids are specified in *xonline.h*.

```
⟨ Specify Services required by the title 17 ⟩ ≡
  const DWORD Services[] = {XONLINE_MATCHMAKING_SERVICE};
  const DWORD dwNumServices = sizeof (Services)/sizeof (Services[0]);
```

This code is used in section 7.

18. The authentication process first authenticates the Xbox. Next, it authenticates each user, and finally authenticates against the requested services specified by *Services* and *dwNumServices* validating that both the users *and* the Xbox have access to them. All three stages are handled by the client APIs, though the title is required to check for errors and handle them appropriately.

```
⟨ Initiate the authentication process 18 ⟩ ≡
  hr = XOnlineLogon(LogonUsers, Services, dwNumServices, Λ, &g_hLogonTask);
  ⟨ Verify XOnlineLogon 19 ⟩;
```

This code is used in section 7.

19. A title should check the return code of the call to *XOnlineLogon*. A return code of `S_OK` indicates that the call succeeded, and the task handle returned should be pumped to complete the authentication process.

```
⟨ Verify XOnlineLogon 19 ⟩ ≡
  switch (hr) {
case S_OK:
  break;
```

See also sections 20 and 21.

This code is used in section 18.

20. *XOnlineLogon* can fail for a number of reasons. If no network connection was detected, `XONLINE_E_LOGON_NO_NETWO` is returned. In this case, the title must give the player the option of accessing the network configuration section of the online dash. For brevity, we simply note the condition and boot to the dash.

```
⟨ Verify XOnlineLogon 19 ⟩ +≡
case XONLINE_E_LOGON_NO_NETWORK_CONNECTION:
  Print(L"No␣network␣connection␣detected.");
  BootToDash(XLD_LAUNCH_DASHBOARD_NETWORK_CONFIGURATION);
  break;
```

21. Here's a catch-all for other values. This should never be reached since we have just enumerated all possible return cases.

```
⟨ Verify XOnlineLogon 19 ⟩ +≡
default:
  assert(FALSE);
  return FALSE;
  }
```

22. System Authentication. After calling *XOnlineLogon*, the next step to repeatedly call *XOnlineTaskContinue* with *g_hLogonTask* as long as `XONLINETASK_S_RUNNING` is returned. This can take up to a minute or more depending on network conditions. To abort the authentication process, a title can call *XOnlineTaskClose*. Note, that in a real title, this would probably appear inside your game loop.

```
⟨ Loop until authentication is complete 22 ⟩ ≡
  Print(L"Signing␣in...");
  do {
    hr = XOnlineTaskContinue(g_hLogonTask);     /* Do a small amount of work */
  } while (hr ≡ XONLINETASK_S_RUNNING);     /* As long as there is work to do */
```

This code is used in section 7.

23. Once the task has completed, a title must check the return value to see if system authentication succeeded. A value of `XONLINE_S_LOGON_CONNECTION_ESTABLISHED` indicates the system authentication was indeed successful.

```
⟨ Verify that Xbox authentication was successful 23 ⟩ ≡
  switch (hr) {
case XONLINE_S_LOGON_CONNECTION_ESTABLISHED: Print(L"Connection␣established");
  break;
```

See also sections 24, 25, 26, 27, 28, and 29.

This code is used in section 7.

24. If, during the authentication process, the Xbox loses its network connection, `XONLINE_E_LOGON_CONNECTION_LOST` will be returned. If this is the case, the title should allow the player the *option* of booting into the network configuration section of the online dash. For brevity, this sample will just boot to the network configuration section of the online dash.

```
⟨ Verify that Xbox authentication was successful 23 ⟩ +≡
case XONLINE_E_LOGON_CONNECTION_LOST: Print(L"Network␣connection␣lost");
  BootToDash(XLD_LAUNCH_DASHBOARD_NETWORK_CONFIGURATION);
  return FALSE;
```

25. The `XONLINE_E_LOGON_CANNOT_ACCESS_SERVICE` error indicates that the console was unable to access the Xbox Live service. In this case, the title should allow the player the *option* of booting into the network configuration section of the online dash. Again, in the intrest of brevity, we just boot to the appropriate section of the dash

```
⟨ Verify that Xbox authentication was successful 23 ⟩ +≡
case XONLINE_E_LOGON_CANNOT_ACCESS_SERVICE:
  UIMsg(L"Your␣Xbox␣console␣cannot␣connect␣to␣Xbox␣Live.\n"
  L"Press␣A␣to␣start␣the␣troubleshooter␣or␣B␣to␣cancel.");
  BootToDash(XLD_LAUNCH_DASHBOARD_NETWORK_CONFIGURATION);
  return FALSE;
```

26. `XONLINE_E_LOGON_UPDATE_REQUIRED` is returned when an updated version of this title is available on the server and gameplay must not continue until the title is updated. The title must allow the user the option of updating the title before continuing. If the user decides to update immediately, the title is required to call *XOnlineTitleUpdate* so that the update is downloaded to the hard drive. The *XOnlineTitleUpdate* function will boot into an *updater application*, which performs the actual update. Once complete, the updated title will be executed. Please note the autoupdate is intended as a means of addressing catastrophic defects or security holes in a shipping title, and is *not* a general purpose update mechanism for adding new features.

```
⟨ Verify that Xbox authentication was successful 23 ⟩ +≡
case XONLINE_E_LOGON_UPDATE_REQUIRED:
  UIMsg(L"A␣required␣update␣is␣available␣for␣the␣XBox␣Live␣Service.\n"
  L"Press␣A␣to␣update␣or␣B␣to␣cancel.␣␣You␣cannot␣connect\n"
  L"to␣Xbox␣Live␣until␣the␣update␣is␣installed.");
  XOnlineTitleUpdate(0);
  assert(FALSE);    /* Should not reach here */
  return FALSE;
```

27. `XONLINE_E_LOGON_INVALID_USER` is returned when one or more users has an unrecognized Gamertag or key. A title should allow the player the *option* of booting into the account management section of the online dash. For brevity, we note the condition and boot into the account management section of the dash.

```
⟨ Verify that Xbox authentication was successful 23 ⟩ +≡
case XONLINE_E_LOGON_INVALID_USER: Print(L"Invalid␣user␣detected.");
  BootToDash(XLD_LAUNCH_DASHBOARD_ACCOUNT_MANAGEMENT);
  assert(FALSE);    /* Should not reach here */
  return FALSE;
```

28. If `XONLINE_E_LOGON_SERVERS_TOO_BUSY` is returned, the Xbox Live service is too busy at the moment. A title should indicate this (using the recommended text passed to *UIMsg*) and give the player the option of trying again. In the interest of brevity, the sample simply displays the recommended message and bails.

```
⟨ Verify that Xbox authentication was successful 23 ⟩ +≡
case XONLINE_E_LOGON_SERVERS_TOO_BUSY: UIMsg(L"The␣Xbox␣Live␣service␣is␣very␣busy.\n"
  L"Press␣A␣to␣try␣again␣or␣press␣B␣to␣cancel.");
  return FALSE;
```

29. For other, unexpected errors, we simply note the error and bail.

```
⟨ Verify that Xbox authentication was successful 23 ⟩ +≡
default:    /* Some other error - title is free to allow access to dash */
  Print(L"Login␣failed␣with␣error␣0x%x", hr);
  return FALSE;
  }
```

30. User Authentication. To check for user authentication errors, we first call *XOnlineGetLogonUsers*. This returns a pointer to an array of **XONLINE_USER** structures. This array is similar the **XONLINE_USER** array we populated and passed into *XOnlineLogon*, but is updated with error status and permission flags for each user.

```
⟨ Verify that user authentication was successful 30 ⟩ ≡
  PXONLINE_USER Users = XOnlineGetLogonUsers( );
  assert(Users);
```

See also section 31.

This code is used in section 7.

31. For each user account assigned to a controller we next check to if that user was successfully signed in. If no user was assigned to a controller, the corresponding **XONLINE_USER** entry in *Users* will be zeroed out. We can test for this by checking to see if the user id *Users*[*i*].*xuid*.*qwUserID* is zero or not (valid user account never have an a zero user id). For each valid user, we also test for various user permissions and make note of them.

```
⟨ Verify that user authentication was successful 30 ⟩ +≡
  for (DWORD i = 0; i < XONLINE_MAX_LOGON_USERS; ++i) {
    if (Users[i].xuid.qwUserID ≠ 0)      /* A valid user */
    {
      DWORD dwUserFlags = Users[i].xuid.dwUserFlags;
      BOOL bGuest = XOnlineIsUserGuest(dwUserFlags);
      ⟨ Verify user authentication 32 ⟩
      ⟨ Check for user permissions 36 ⟩
    }
  }
```

32. The User array returned by *XOnlineGetLogonUsers* has the *hr* field of each element set with a status code indicating whether or not authentication for that user succeeded. If *hr* is set to `S_OK`, then the user (or guest) was successfully signed in.

```
⟨ Verify user authentication 32 ⟩ ≡
  switch (Users[i].hr) {
case S_OK:
  if (bGuest) Print(L"Guest␣%d␣of␣%S␣signed␣in", XOnlineUserGuestNumber(dwUserFlags),
          Users[i].szGamertag);
  else Print(L"%S␣signed␣in", Users[i].szGamertag);
  break;
```

See also sections 33, 34, and 35.

This code is used in section 31.

33. If *hr* is set to `XONLINE_S_LOGON_USER_HAS_MESSAGE`, the user has a message from the Xbox Live Service. The title must allow the *option* of booting into the account management section of the online dash in order view the messages. For brevity, we note the condition, display the recommened user interface mesage and boot to the account management section of the dash.

```
⟨ Verify user authentication 32 ⟩ +≡
case XONLINE_S_LOGON_USER_HAS_MESSAGE: Print(L"%S␣signed␣in,␣and␣has␣messages",
  Users[i].szGamertag);
  UIMsg(L"You␣have␣a␣new␣Xbox␣Live␣message.\n"
  L"Press␣A␣to␣read␣it␣now,␣or␣B␣to␣read␣later.");
  BootToDash(XLD_LAUNCH_DASHBOARD_ACCOUNT_MANAGEMENT);
  break;
```

34. If *hr* is set to `XONLINE_E_LOGON_USER_ACCOUNT_REQUIRES_MANAGEMENT`, the authentication failed, and the user account requires management by user before that user can sign in. For brevity, we note the condition, display the recommened user interface mesage and boot to the account management section of the dash.

```
⟨ Verify user authentication 32 ⟩ +≡
case XONLINE_E_LOGON_USER_ACCOUNT_REQUIRES_MANAGEMENT:
  Print(L"This␣%S␣account␣requires␣management", Users[i].szGamertag);
  UIMsg(L"You␣have␣an␣important␣message␣from␣Xbox␣Live.\n"
  L"Press␣A␣to␣read␣the␣message.");
  BootToDash(XLD_LAUNCH_DASHBOARD_ACCOUNT_MANAGEMENT);
  return FALSE;
```

35. Finally, here's the catch-all case, which should never be reached since we have enumerated all possible status codes.

```
⟨ Verify user authentication 32 ⟩ +≡
default:    /* Should never happen */
  assert(FALSE);
  return FALSE;
  }
```

36. User permissions. The users returned by *XOnlineGetLogonUsers* also has the *dwUserFlags* member of each user `XUID` updated with the latest permission bits for that user. Here we examine those flags checking for various types of user permissions.

```
⟨ Check for user permissions 36 ⟩ ≡
  ⟨ Check if user is allowed to use voice 37 ⟩
  ⟨ Check if user is allowed to purchase 38 ⟩
  ⟨ Check if user is nickname banned 39 ⟩
```

This code is used in section 31.

37. A title can check if user is allowed to use voice by using the *XOnlineIsUserVoiceAllowed* macro.

```
⟨ Check if user is allowed to use voice 37 ⟩ ≡
  if (XOnlineIsUserVoiceAllowed(dwUserFlags))
    Print(L"␣␣␣␣%S␣is␣allowed␣to␣use␣voice", Users[i].szGamertag);
```

This code is used in section 36.

38. The *XOnlineIsUserPurchaseAllowed* macro is used to test if a user is allowed to make purchases.

```
⟨ Check if user is allowed to purchase 38 ⟩ ≡
  if (XOnlineIsUserPurchaseAllowed(dwUserFlags))
    Print(L"␣␣␣␣%S␣is␣allowed␣to␣purchase", Users[i].szGamertag);
```

This code is used in section 36.

39. Finally, *XOnlineIsUserNicknameAllowed* macro is used to test if a user is nickname banned.

```
⟨ Check if user is nickname banned 39 ⟩ ≡
  if (bGuest) {
    if (¬XOnlineIsUserNicknameAllowed(dwUserFlags))
      Print(L"␣␣␣␣Guest␣%d␣of␣%S␣is␣nickname␣banned", XOnlineUserGuestNumber(dwUserFlags),
          Users[i].szGamertag);
  }
  else {
    if (¬XOnlineIsUserNicknameAllowed(dwUserFlags))
      Print(L"␣␣␣␣%S␣is␣nickname␣banned", Users[i].szGamertag);
  }
```

This code is used in section 36.

40. Service Authentication. Check that the requested services are available. For each service that was requested, we call *XOnlineGetServiceInfo* which returns the connection status for that service

```
⟨ Verify that service authentication was successful 40 ⟩ ≡
  for (DWORD i = 0; i < dwNumServices; ++i) {
    hr = XOnlineGetServiceInfo(Services[i], Λ);
    ⟨ Verify service availability 41 ⟩
  }
```

This code is used in section 7.

41. If the return value of *XOnlineGetServiceInfo* was `S_OK`, then a successful connection to the service was made.

```
⟨ Verify service availability 41 ⟩ ≡
  switch (hr) {
case S_OK:
  Print(L"Service␣%lu␣Available", Services[i]);
  break;
```

See also sections 42, 43, and 44.

This code is used in section 40.

42. If the return value was `XONLINE_E_LOGON_SERVICE_NOT_AUTHORIZED`, then one or more of the logged on users was not authorized to use the service. An example of this might be a billing service lockout.

```
⟨ Verify service availability 41 ⟩ +≡
case XONLINE_E_LOGON_SERVICE_NOT_AUTHORIZED:
  Print(L"Access␣to␣service␣%lu␣is␣denied", Services[i]);
  return FALSE;
```

43. A return value of `XONLINE_E_LOGON_SERVICE_TEMPORARILY_UNAVAILABLE` means the service is temporarily unavailable. The title can proceed with other online functionality that does not use this service. If the user attempts to perform functions related to this service, the title should inform the user that the service is temporarily unavailable.

```
⟨ Verify service availability 41 ⟩ +≡
case XONLINE_E_LOGON_SERVICE_TEMPORARILY_UNAVAILABLE:
  Print(L"Service␣%lu␣is␣unavailable", Services[i]);
  return FALSE;
```

44. Finally, we treat any other return value as an error.

```
⟨ Verify service availability 41 ⟩ +≡
default:
  Print(L"Error␣0x%x␣signing␣onto␣service␣%lu", hr, Services[i]);
  return FALSE;
  }
```

45. Setting Presence. For each user (except guests) set their online notification state so they are visible to their online friends. The code calls the *XOnlineNotificationSetState* function passing a controller index i, and a set of online notification flags to do this. A real title would check for the voice peripheral and specify the `XONLINE_FRIENDSTATE_FLAG_VOICE` flag if present. It would also update the notification state as the user joined or left a game session or when the voice peripheral was inserted or removed.

```
⟨ Set Presence 45 ⟩ ≡
  for (DWORD i = 0; i < XONLINE_MAX_LOGON_USERS; ++i) {
    if (Users[i].xuid.qwUserID ≠ 0 ∧ ¬XOnlineIsUserGuest(Users[i].xuid.dwUserFlags)) {
      hr = XOnlineNotificationSetState(i,      /* Controller index */
      XONLINE_FRIENDSTATE_FLAG_ONLINE, XNKID( ), 0, Λ);
      assert(SUCCEEDED(hr));
    }
  }
```

This code is used in section 7.

46. Logon Task Processing in a Game Loop. In addition to performing other game related tasks, such as rendering, the title will need to perform online task processing. For demonstration purposes, our game loop will run for about 15 seconds.

```
VOID GameLoop( )
{   /* Staring time */
  DWORD dwTickStart = GetTickCount( );
  const DWORD dwDuration = 15000;   /* 15 seconds */
  while ((GetTickCount( ) − dwTickStart) < dwDuration)
  {   /* Perform game related frame update and rendering */
    ⟨ Logon Task Processing in a Game Loop 47 ⟩
  }
}
```

47. A title must service the logon task, by calling *XOnlineTaskContinue* with *g_hLogonTask*, inside of its game loop. Failure to pump the task in a timely manner will result in automatic signoff from the system.

```
⟨ Logon Task Processing in a Game Loop 47 ⟩ ≡
  HRESULT hr = XOnlineTaskContinue(g_hLogonTask);
  ⟨ Verify Logon Task 48 ⟩
```

This code is used in section 46.

48. The title should always check the value returned from *XOnlineTaskContinue* for any errors. If `XONLINE_S_LOGON_CONNECTION_ESTABLISHED` is returned, the logon task is still established and there are no errors.

```
⟨ Verify Logon Task 48 ⟩ ≡
  switch (hr) {
case XONLINE_S_LOGON_CONNECTION_ESTABLISHED: break;
```

See also sections 49, 50, and 51.

This code is used in section 47.

49. `XONLINE_E_LOGON_CONNECTION_LOST` is returned when the connection to the XBox Live service has been lost. A title will need to call *XOnlineLogon* once again to re-connect (the old task handle should be closed using *XOnlineTaskClose* first)

```
⟨ Verify Logon Task 48 ⟩ +≡
case XONLINE_E_LOGON_CONNECTION_LOST:
  Print("Connection␣Lost.");
  return;
```

50. A title is required to check for the case where a player has been signed out because that same account has been signed on another Xbox. This is indicated by the `XONLINE_E_LOGON_KICKED_BY_DUPLICATE_LOGON` error code.

```
⟨ Verify Logon Task 48 ⟩ +≡
case XONLINE_E_LOGON_KICKED_BY_DUPLICATE_LOGON:
  UIMsg(L"You␣were␣signed␣out␣of␣Xbox␣Live␣because␣another\n"
  L"person␣signed␣on␣using␣your␣account.\n"
  L"Press␣A␣to␣continue.");
  return;
```

51. Finally we add a catch-all case, which should never occurr as we have enumerated all possible return values.

```
⟨ Verify Logon Task 48 ⟩ +≡
default:
  assert(FALSE);
  return;
  }
```

52. Signing off of the XBox Live Service. A title signs off users by calling *XOnlineTaskClose* on the task handle returned by *XOnlineLogon*. Another situation in which users are signed off is if the Xbox Live Service realizes the task handle returned by *XOnlineLogon* is not being serviced by the title (e.g. the user turned the console off).

⟨ Sign off of the Xbox Live Service 52 ⟩ ≡
  XOnlineTaskClose(*g_hLogonTask*);

This code is used in section 3.

53. XBox Device Handling. The XInitDevices function initializes the peripheral control software on the console, and also allows the title to specify the maximum number of devices of each device type that will ever be used simultaneously. This function must be called before any peripherals (controllers, memory cards, voice units, and so on) are used or enumerated by the title. We specify zero and NULL so that the maximum limits for each device type used will apply.

⟨ Initialize Devices 53 ⟩ ≡
```
   XInitDevices(0, Λ);
```
This code is used in section 3.

54. Before we can enumerate user accounts on any attached Memory Units, we must first allow them sufficient time to mount. Depending on the number of Memory Units which have been inserted this may take a few seconds. The *XGetDeviceEnumerationStatus* function should be called in busy loop until it returns a value other than `XDEVICE_ENUMERATION_BUSY`.

⟨ Wait for Memory Units to mount 54 ⟩ ≡
```
   while (XGetDeviceEnumerationStatus() ≡ XDEVICE_ENUMERATION_BUSY) { }
```
This code is used in section 3.

55. Miscellaneous Functions. The *Print* sends formatted text to *debug output* using the *OutputDebugString* function. The function accepts a format string and a variable number of arguments. It works much like *printf* .

```
VOID __cdecl Print(const WCHAR*strFormat, ...)
{
  const int MAX_OUTPUT_STR = 80;
  WCHARstrBuffer[MAX_OUTPUT_STR];
  va_list pArglist;
  va_start(pArglist, strFormat);
  INTiChars = wvsprintfW(strBuffer, strFormat, pArglist);
  assert(iChars < MAX_OUTPUT_STR);
  OutputDebugStringW(L"\n***␣SimpleAuth:␣");
  OutputDebugStringW(strBuffer);
  OutputDebugStringW(L"\n\n");
  (VOID) iChars;     /* avoid compiler warning */
  va_end(pArglist);
}
```

56. The *UIMsg* function is used for displaying recommended user interface messages. Any call to *UIMsg* in this sample is meant to specify text which a real title should use when providing information to a player.

```
VOID UIMsg(const WCHAR*strText)
{
  OutputDebugStringW(L"\n***␣SimpleAuth:␣UI␣Message:\n");
  OutputDebugStringW(strText);
  OutputDebugStringW(L"\n");
}
```

57. The *BootToDash* function will boot to a specified section in the dash. It calls *XLaunchNewImage*, which never returns.

```
VOID BootToDash(DWORD dwReason)
{
  LD_LAUNCH_DASHBOARDld;
  ZeroMemory(&ld, sizeof (ld));
  ld.dwReason = dwReason;
  XLaunchNewImage(Λ, PLAUNCH_DATA(&ld));
  assert(FALSE);     /* Unreachable */
}
```

58. Function Prototypes. Here are the prototypes for the functions in this sample.

⟨ Function Prototypes 58 ⟩ ≡

```
  BOOL SignIn( );
  VOID GameLoop( );
  VOID __cdecl Print(const WCHAR *strFormat, ...);
  VOID UIMsg(const WCHAR *strText);
  VOID BootToDash(DWORD dwReason);
```

This code is used in section 3.

59. Index.

Xbox Authentication Sample