Email for iOS

`Email` provides functionality for configuring and launching the native mail composer and receiving responses.

Core Functionality

Member	Description
`static void Send(MessageConfig config, Action<Result> callback)`	Sends an email using the specified configuration. A native mail window will open, configured according to your settings.
`static bool IsSendAvailable { get; }`	Indicates whether sending emails is available on the current device.

Message

The `Message` class contains all the configuration options for composing an email.

Member	Description
`List<string> Recipients { get; set; }`	The primary recipients of the email.
`List<string> Cc { get; set; }`	The CC (carbon copy) recipients of the email.
`List<string> Bcc { get; set; }`	The BCC (blind carbon copy) recipients of the email.
`string Subject { get; set; }`	The subject of the email.
`string Body { get; set; }`	The body of the email.
`bool IsHtml { get; set; }`	Indicates whether the email body is HTML.
`string AttachmentPath { get; set; }`	The file path of the email attachment.
`string AttachmentMimeType { get; set; }`	The MIME type of the email attachment.
`string AttachmentFileName { get; set; }`	The file name of the email attachment.

Result Enum

The `Result` enumeration provides information about the outcome of the email operation.

Member	Description
`None`	No result has been determined yet.
`Cancelled`	The email operation was cancelled by the user.
`Saved`	The email was saved as a draft.
`Sent`	The email was successfully sent.
`Failed`	The email operation failed.
`Error`	An error occurred during the email operation.
`Unavailable`	Sending emails is unavailable.

Usage Example

```
// Using callback
Email.Send(new Message
{
    Recipients = new List<string> { "recipient@example.com" },
    Cc = new List<string> { "cc@example.com" },
    Bcc = new List<string> { "bcc@example.com" },
    Subject = "Test Subject",
    Body = "This is a test email.",
    IsHtml = false,
    // Optional attachment
    AttachmentPath = Path.Combine(Application.persistentDataPath, "document.pdf"),
    AttachmentMimeType = "application/pdf",
    AttachmentFileName = "document.pdf"
}, result =>
{
    Debug.Log($"Email result: {result}");

    switch(result)
    {
        case Result.Sent:
            Debug.Log("Email was sent successfully! (iOS)");
            break;
        case Result.Cancelled:
            Debug.Log("User cancelled sending the email (iOS)");
            break;
        case Result.Saved:
            Debug.Log("Email was saved as draft (iOS)");
            break;
        case Result.Completed:
            Debug.Log("Email operation completed (Android)");
            break;
        // Handle other results as needed
    }
});

// Using async/await
async void SendEmailAsync()
{
    var result = await Email.SendAsync(new Message
    {
        Recipients = new List<string> { "recipient@example.com" },
        Subject = "Async Email Test",
        Body = "This email was sent using async/await."
    });

    Debug.Log($"Async email result: {result}");
}
```

Attachment Guidelines

iOS

On iOS, attachments can be from any location accessible by your app, including:

- App's internal storage (`Application.persistentDataPath`)
- App's bundle resources (`Application.streamingAssetsPath`)
- Files accessible through Unity's file handling APIs

Note: While iOS theoretically supports accessing files from more locations, for cross-platform compatibility we recommend using `Application.persistentDataPath` for storing attachments, just like on Android.

Checking Email Availability

Before attempting to send an email, you can check if email functionality is available:

```
// Then check if email sending is available
if (Email.IsSendAvailable)
{
    // Show email UI and functionality
    ShowEmailButton();
}
else
{
    // Hide or disable email functionality
    HideEmailButton();
    Debug.LogWarning("Email functionality not available on this device");
}
```