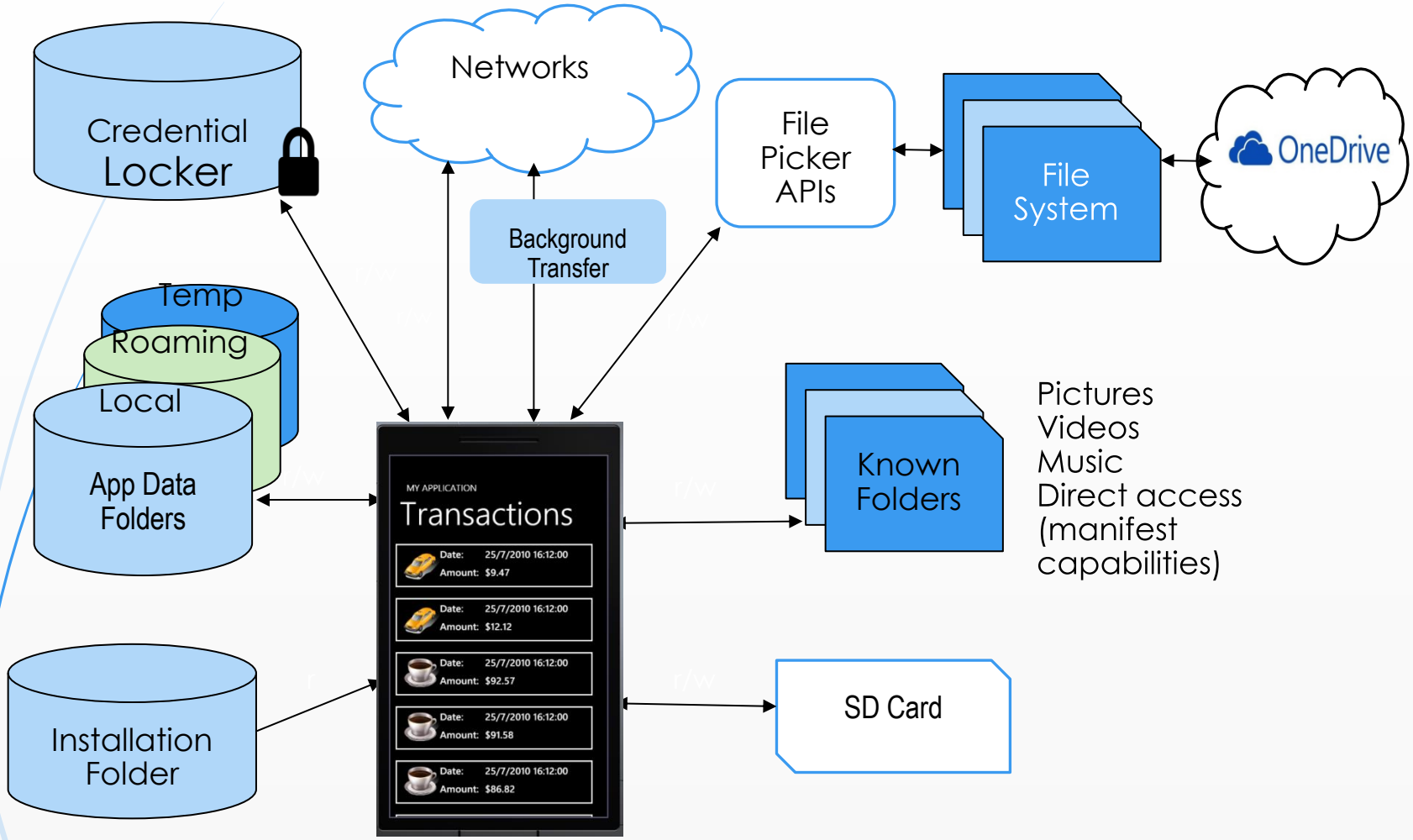
Module

Data Storage

Table of contents

Data Storage Locations



Data Storage Locations

- When an app is installed,
  - The system gives it its own per-user data store for app data such as settings and files
  - The programmer doesn't need to know where or how this data exists, because the system is responsible for managing the physical storage
  - He can use the app data API to work with the app data

Data Storage Locations

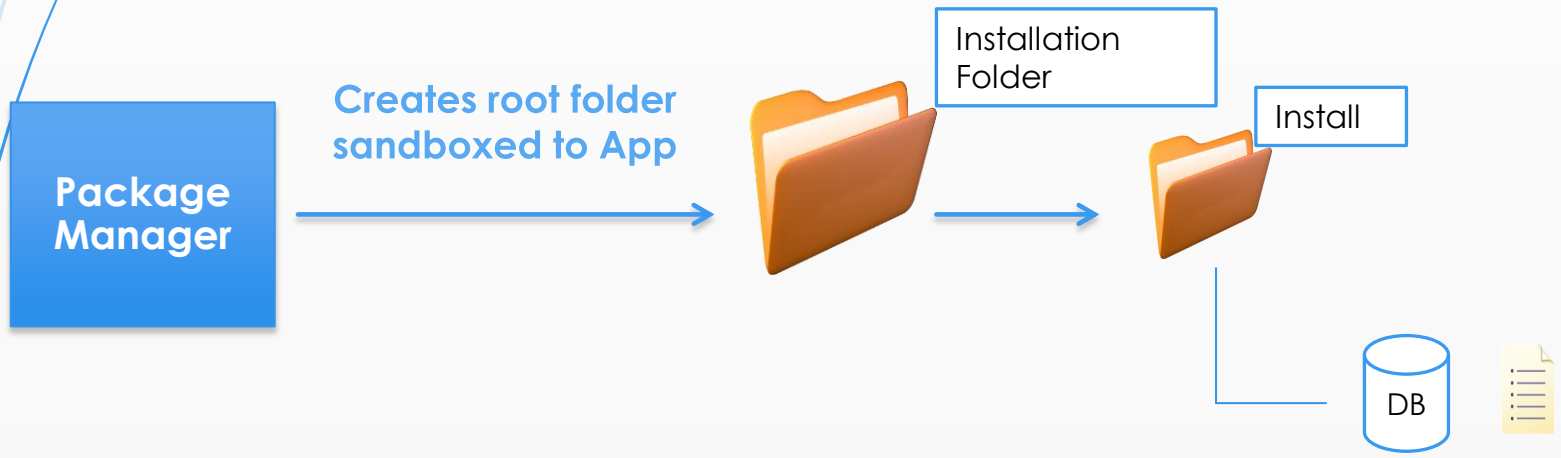
- Data store
  - Local (Default storage location)
    - Exists on the current device and is backed up in the cloud
  - Roaming
    - Exists on all devices on which the user has installed the app
  - Temporary
    - Can be cleaned up by the system in the event of a low-storage situation
  - LocalCache
    - Persistent data that exists only on the current device
    - Identical to Local but no backup

Data Storage Locations

- If the app is removed, the data stores are deleted
- Optionally, the developer can put a number version for the app data
  - Possibility of a future version of the app that changes the format of its app data without causing compatibility problems with the previous version
  - The app checks the version of the app data in the data stores
    - If the number version is less than the number version the app expects, the app should update the app data to the new format and update the version

Installation Folder

- All app files are installed by the Package Manager into the *Installation Folder*
  - Read-only access from the app
  - Read-only reference database/file

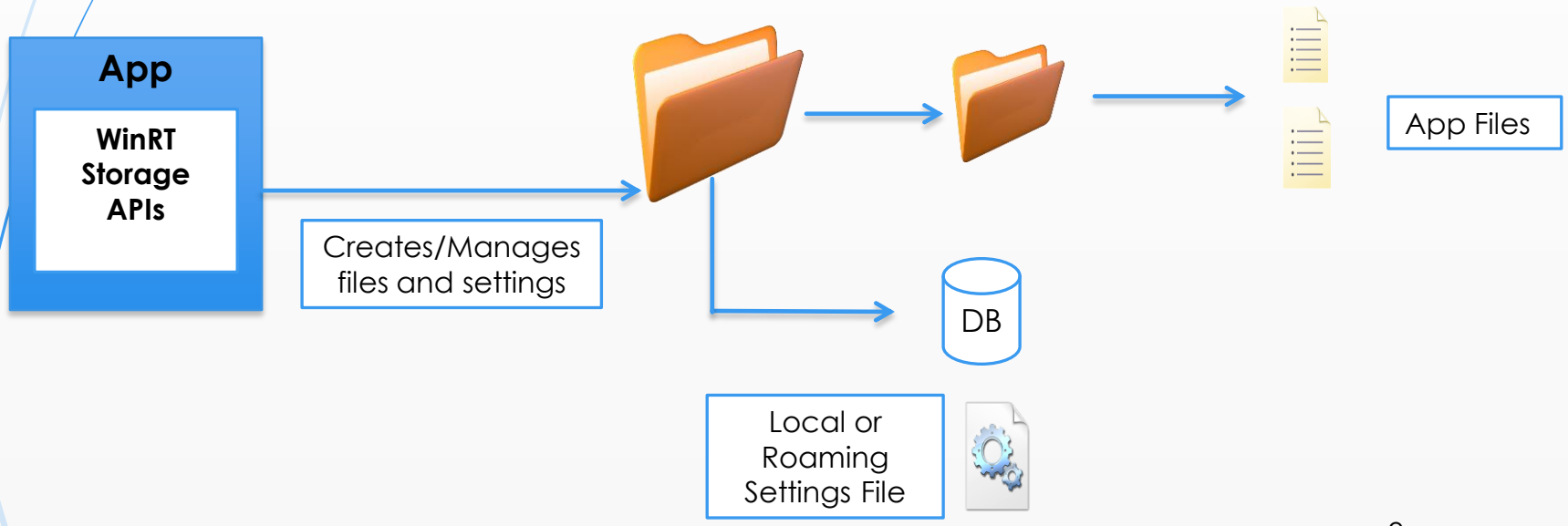


Installation Folder

- When publishing an update for the app on the Windows Store, the Install folder of the app is replaced
- Read-only database
  - Structured reference data for the app
  - Data that may change when each app is updated but not editable through the app
    - E.g., a definition of words dictionary

App Data Storage

- Data can be stored by the app in the App Data folders (Roaming, Local and Temp)



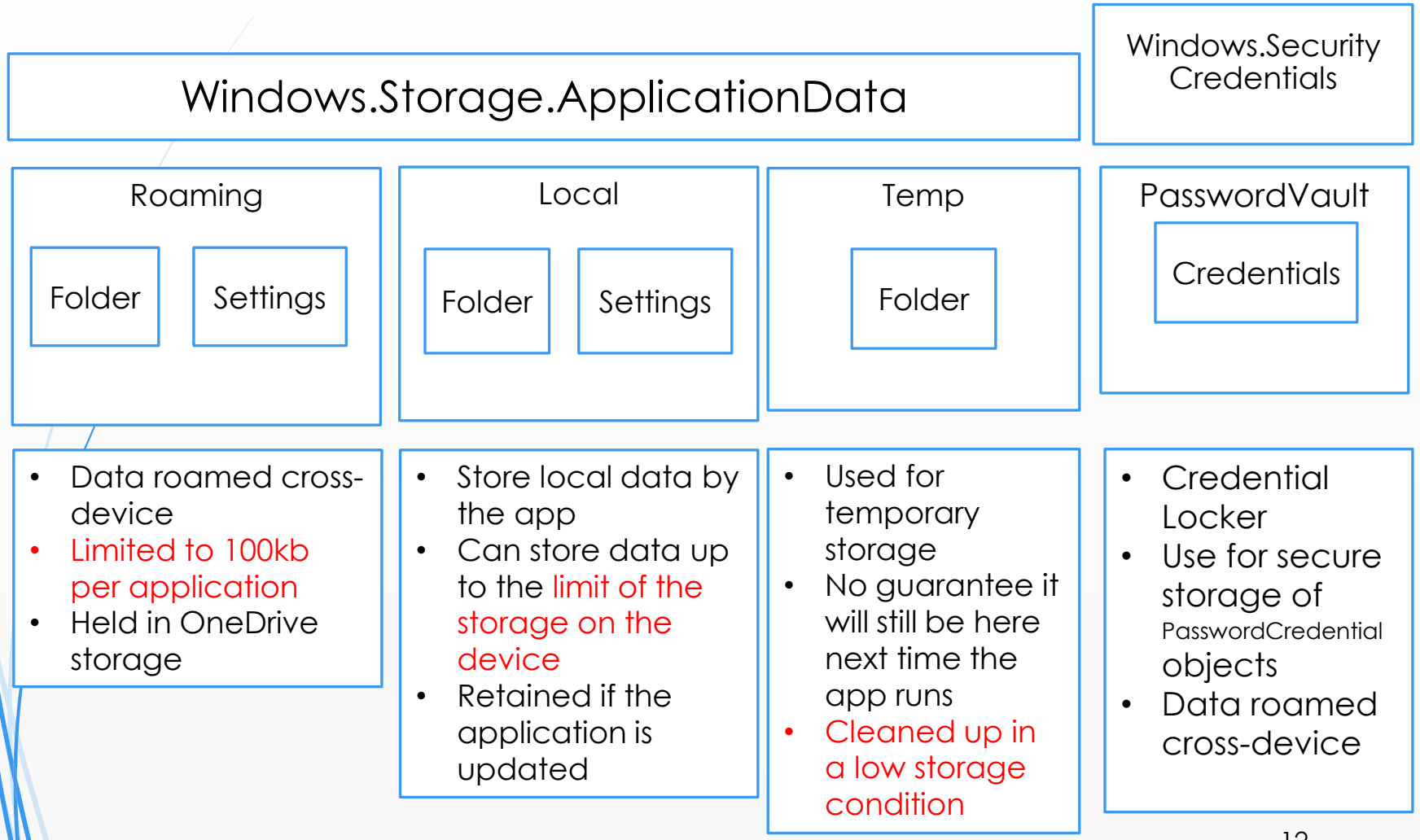
App Data Storage

- Storing data in a WP app data model
  - Multiple options
    - Roaming
    - Local
    - LocalCache
    - Temporary
  - Data accessible through the *Windows.Storage.ApplicationData* class
    - Provides access to the app data stores

App Data Storage

- Roaming
  - Stored data eligible for roaming synchronization between the user's devices
    - Including synchronization between Windows and Windows Phone for universal apps with a shared identity
  - Roaming data may also be backed up under conditions
    - E.g., when the user has disabled roaming in order to capture the entirety of the app state

App Data Storage



App Data Storage

- *Windows.Storage.ApplicationData*
  - Provides access to the
    - 3 storage folders in the app data stores
    - And the app settings containers
- StorageFolder
    - Represents a folder
    - Is used to access the folder and its contents (files)

```
Windows.Storage.StorageFolder roamingFolder =
Windows.Storage.ApplicationData.Current.RoamingFolder;
Windows.Storage.StorageFolder localFolder = Windows.Storage.ApplicationData.Current.LocalFolder;
Windows.Storage.StorageFolder temporaryFolder =
Windows.Storage.ApplicationData.Current.TemporaryFolder;
```

App Data Storage

- ApplicationDataContainer
  - Represents a container for app settings
  - To support creating, deleting, enumerating and traversing the container hierarchy

```
Windows.Storage.ApplicationDataContainer localSettings =
                Windows.Storage.ApplicationData.Current.LocalSettings;

Windows.Storage.ApplicationDataContainer roamingSettings =
                Windows.Storage.ApplicationData.Current.RoamingSettings;
```

Local and Roaming Settings

- Declaration for LocalSettings

```
var appData = Windows.Storage.ApplicationData.Current;
var localSettings = appData.LocalSettings;
```

- To create a setting

```
localSettings.Values["mySetting"] = "Windows 10";
```

- To read a setting

```
Object value = localSettings.Values["mySetting"];
if (value != null) ….
```

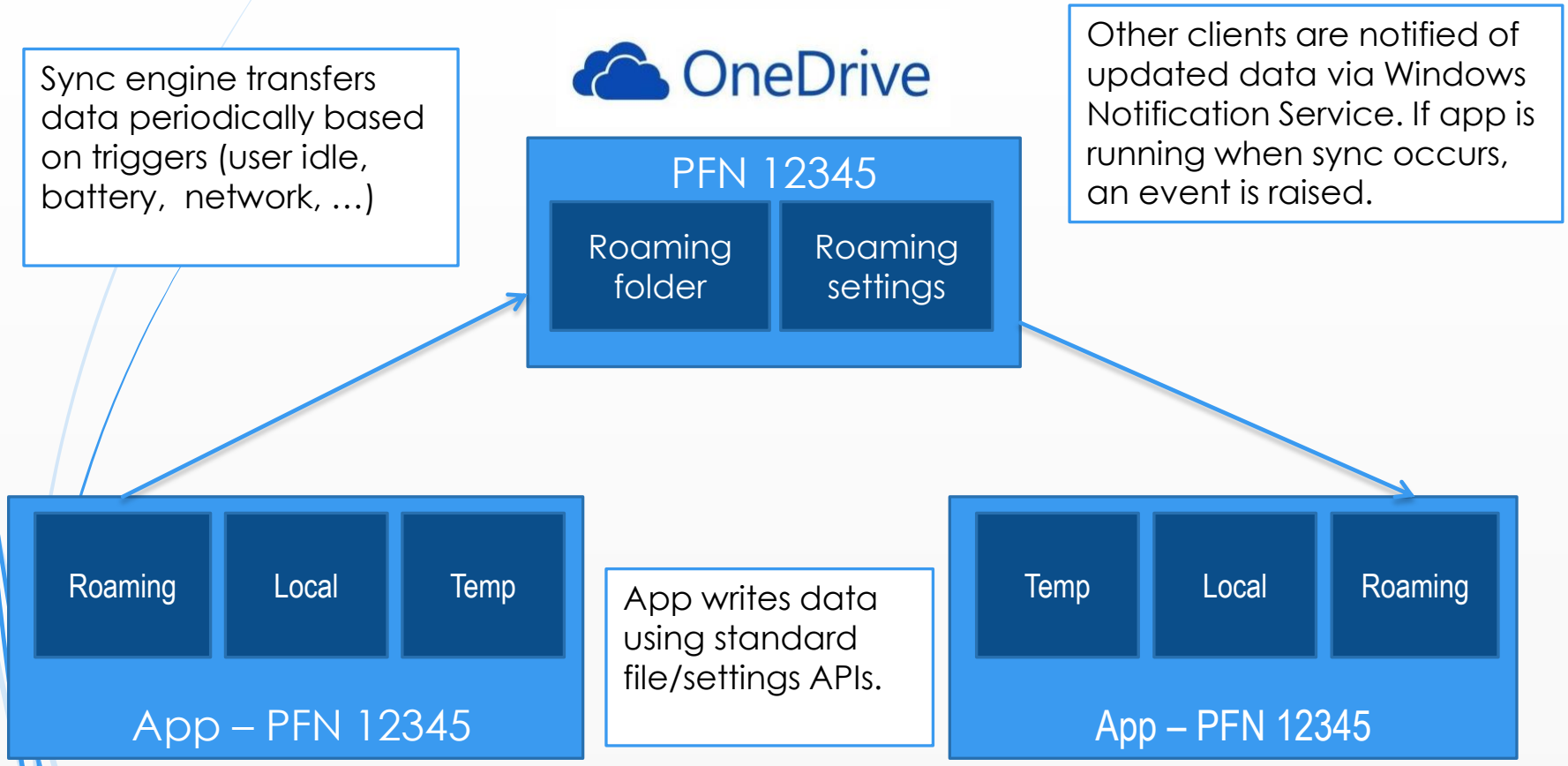
- To delete a setting

```
localSettings.Values.Remove("mySetting");
```

Local and Roaming Settings

- Roaming settings
  - Same way as local settings
  - If a user installs the app on multiple devices, all the devices can share the same settings information
    - Changes on one device are reflected on all the other devices
  - The synchronisation takes place in the background

Roaming


Sync engine transfers data periodically based on triggers (user idle, battery, network, …)
OneDrive
PFN 12345
Roaming folder
Roaming settings
Other clients are notified of updated data via Windows Notification Service. If app is running when sync occurs, an event is raised.
Roaming
Local
Temp
App – PFN 12345
App writes data using standard file/settings APIs.
Temp
Local
Roaming
App – PFN 12345

Roaming

- Dictionary into which an app can save data
  - Persistence of the data on the device
  - Shared with other devices
  - On Windows 8.1, special *HighPriority* setting; not on Windows Phone
  - The key name will be reflected across multiple devices (including Windows 8.1)

Roaming

- E.g.,

```
private void nameStudent_TextChanged(object sender, TextChangedEventArgs e)
{
    var appData =  Windows.Storage.ApplicationData.Current;
    var roamingSettings =  appData.RoamingSettings;
    roamingSettings.Values["nameStudentText"] = nameStudent.Text;
}
```

```
var appData =  Windows.Storage.ApplicationData.Current;
var roamingSettings =  appData.RoamingSettings;

if (roamingSettings.Values.ContainsKey("nameStudentText"))
{
    nameStudent.Text = roamingSettings.Values["nameStudentText"].ToString();
}
```

Roaming

- When the roaming data changes, the user can receive notification
  - *DataChanged* event
    - Only fired if the app is active at the time of change

```
Windows.Storage.ApplicationData.Current.DataChanged += Current_DataChanged;
...

void Current_DataChanged(ApplicationData sender, object args)
{
    // Refresh the settings
}
```

  - The developer should still load up all the data when the app starts

Addressing Storage Locations

File Type/ API	Installation Folder	App data folder	Example
File access using Windows.Storage API via URIs	ms-appx:///	ms-appdata:///local/ ms-appdata:///roaming/ ms-appdata:///temp/	var file = await Windows.StorageFile.GetFileFromApplicationUriAsync(new Uri (ms-appdate:///local/AppConfigSettings.xml));
File access using Windows.Storage API with StorageFolder references	Windows. ApplicationModel.Package.Current. InstalledLocation	Windows.Storage. ApplicationData. Current .LocalFolder / .RoamingFolder / .TempFolder	var localFolder = Windows.Storage.ApplicationData.Current.LocalFolder; Windows.Storage.StorageFile storageFile = await localFolder.GetFileAsync("…");

File Writing/Reading

- Async operations
- To write (to create) a file
  - *FileIO* API
  - Stream
    - It handles the difference between one type of storage (e.g. memory) and another one (e.g. network or file system)
    - It is used all over WinRT

File Writing/Reading

- To write in a file (to create)

```
private async void WriteTextToLocalStorageFile(string filename, string text)
{
    var localFolder = Windows.Storage.ApplicationData.Current.LocalFolder;
    StorageFile  storageFile =
        await localFolder.CreateFileAsync(filename, CreationCollisionOption.ReplaceExisting);
    await FileIO.WriteTextAsync(storageFile, text);
}
```

- To read a file

```
private async Task<string> ReadTextFromLocalStorageFile(string filename)
{
    var localFolder = Windows.Storage.ApplicationData.Current.LocalFolder;
    StorageFile storageFile = await localFolder.GetFileAsync(filename);
    string result = await FileIO.ReadTextAsync(storageFile);
    return result;
}
```

File Writing/Reading

- To write in a file (to create) with a stream

```
private async void WriteTextToLocalStorageFileWithStream(string filename, string text)

{
   var  localFolder = Windows.Storage.ApplicationData.Current.LocalFolder;
   StorageFile storageFile = await localFolder.CreateFileAsync(filename);

    using (StorageStreamTransaction transaction = await storageFile.OpenTransactedWriteAsync())
      {
         using (DataWriter dataWriter = new DataWriter(transaction.Stream))
         {
            dataWriter.WriteString(text);
            // reset stream size to override the file
            transaction.Stream.Size = await dataWriter.StoreAsync();
            await transaction.CommitAsync();
         }
      }
}
```

XML / JSON Serializers

- Serialization
  - A programming technique that converts an object to a sequence of bytes
- Serialized data
  - Is suitable for storage in files and databases
  - Can be sent to other computer systems across network protocols
- To store objects
  - Serialization data in XML or JSON formats
  - Data stored or transferred as text
- Deserialization

XML / JSON Serializers

- E.g., for JSON,

```
[DataContract]
public class Student {
    [DataMember]
     public string name;
     [DataMember]
     public int age;
}
```

```
Student myStudent = new Student(…);
MemoryStream stream = new MemoryStream();
DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(Student));
serializer.WriteObject(stream, myStudent);

Student verification = (Student)serializer.ReadObject(stream);
```

Compression

- To save and load data with compression
  - The compression stream can be used as any other stream, but it compresses the data as it is transferred
- Different algorithms
  - Lzms
  - Mszip
  - Xpress
  - XpressHuff

Credential Locker

- Simplification for the management of
  - user credentials
  - their encrypted storage on the device where the app is running
- Also roaming of the credentials between devices (with the user Microsoft account)
- Information stored in the Credential Locker per user (not shared between apps)

KnownFolders

- Files in *KnownFolders* are visible for all apps (that have registered the proper capabilities)
- *Windows.Storage.KnownFolders* class
  - Provides access to common locations that contain user content
    - This includes content from a user's local libraries (such as Documents, Pictures, Music, and Videos), removable devices and media server devices

KnownFolders

- Rather than searching through all the possible different locations on the device for a particular type of file,
  - The program can request a single list of all the files
    - This includes files on the SD card (if inserted)
- FileOpenPicker API is an alternative to allow users to select a file in these folders

KnownFolders

- E.g.,

```
private async void LoadVideo() {
    …
    IReadOnlyList<Windows.Storage.StorageFile> resultsLibrary = await
                                        Windows.Storage.KnownFolders.VideosLibrary.GetFilesAsync();

     MessageBlock.Text += "Play video: " + resultsLibrary[0].Name + "\n";

     Windows.Storage.Streams.IRandomAccessStream videoStream = await
                                resultsLibrary[0].OpenAsync(Windows.Storage.FileAccessMode.Read);

      VideoPlayer.SetSource(videoStream, resultsLibrary[0].FileType); VideoPlayer.Play();
      …
}
// http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh465191.aspx
```