Ứng dụng truyền dữ liệu thông qua DataPackage.

WinRT định nghĩa lớp Windows.ApplicationModel.DataTransfer.DataPackage. Ứng dụng nguồn tạo đối tượng DataPackage và khởi tạo với dữ liệu mà ứng dụng muốn chia sẻ với ứng dụng đích. Lớp DataPackage có định nghĩa như sau:

```
public sealed class DataPackage {
   public DataPackage();   // Tạo một đối tượng DataPackage mới

   // Sử dụng để đặt các thuộc tính (ApplicationName, Title, Description,
Thumbnail, v.v.)
   public DataPackagePropertySet Properties { get; }

   // Tùy chọn: Dùng để chỉ định lý do chia sẻ (None, Copy, Move, Link)
   public DataPackageOperation RequestedOperation { get; set; }

   // Gọi SetData nhiều lần để thêm nội dung giống nhau ở các định dạng khác
nhau
   public void SetData(String formatId, Object value);

   // Các phương thức khác...
}
```

Giải thích: Ứng dụng nguồn sử dụng phương thức `SetData` để thêm dữ liệu vào đối tượng DataPackage với các định dạng khác nhau. Ví dụ, một đoạn văn bản HTML có thể được thêm dưới dạng HTML, RTF, và văn bản thuần túy. Ứng dụng đích sẽ chọn định dạng tốt nhất mà nó hỗ trợ từ đối tượng này.

Ứng dụng truyền dữ liệu thông qua DataPackage

WinRT định nghĩa lớp `Windows.ApplicationModel.DataTransfer.DataPackage`. Ứng dụng nguồn tạo một đối tượng `DataPackage` và khởi tạo nó với dữ liệu mà ứng dụng sẵn sàng chia sẻ với ứng dụng đích. Lớp `DataPackage` được định nghĩa như sau:

```
public sealed class DataPackage {
   public DataPackage();   // Tạo một đối tượng DataPackage mới

   // Dùng để thiết lập các thuộc tính (ApplicationName, Title, Description,
Thumbnail, v.v.)
   public DataPackagePropertySet Properties { get; }

   // Tùy chọn: Dùng để chỉ định lý do chia sẻ gói dữ liệu (None, Copy, Move,
Link)
   public DataPackageOperation RequestedOperation { get; set; }

   // Gọi SetData nhiều lần để thêm cùng một nội dung dưới các định dạng khác
nhau
   public void SetData(String formatId, Object value);
```

```
   // Các phương thức này là wrapper an toàn về kiểu dữ liệu cho SetData dành
cho các kiểu phổ biến:
   public void SetText(String value);
   public void SetRtf(String value);
   public void SetHtmlFormat(String value);  // Xem lớp HtmlFormatHelper của
WinRT
   public void SetBitmap(RandomAccessStreamReference value);
   public void SetApplicationLink(Uri value);
   public void SetWebLink(Uri value);
   public void SetStorageItems(IEnumerable<IStorageItem> value);
   public void SetStorageItems(IEnumerable<IStorageItem> value, Boolean
readOnly);

   // Khi thêm HTML có tham chiếu đến nội dung mà ứng dụng đích không thể
truy cập,
   // thêm từng URI và RandomAccessStreamReference vào ResourceMap để ứng
dụng đích có thể
   // truy cập nội dung.
   public IDictionary<String, RandomAccessStreamReference> ResourceMap
{ get; }

   // Tùy chọn: Được kích hoạt sau khi ứng dụng đích dán dữ liệu (ứng dụng
nguồn có thể xóa nội dung nếu muốn)
   public event TypedEventHandler<DataPackage, OperationCompletedEventArgs>
OperationCompleted;

   // Tùy chọn: Được kích hoạt sau khi DataPackage bị GC'd. Điều này cho phép
ứng dụng nguồn
   // xóa nội dung (như tệp tạm thời) nếu muốn:
   public event TypedEventHandler<DataPackage, Object> Destroyed;

   // Thiết lập một delegate để xử lý yêu cầu từ ứng dụng đích
   public void SetDataProvider(String formatId, DataProviderHandler
delayRenderer);

   // Trả về phiên bản chỉ đọc của đối tượng DataPackage (đây là đối tượng mà
ứng dụng đích nhận được)
   public DataPackageView GetView();
}
```

Sau khi ứng dụng nguồn tạo một thể hiện của lớp này, nó đặt một số thuộc tính `DataPackagePropertySet` mô tả dữ liệu (ApplicationName, Title, Description, v.v.). Nếu dữ liệu được chia sẻ qua clipboard, ứng dụng có thể thiết lập thuộc tính `DataPackageOperation`. Tiếp theo, nội dung sẽ được thêm vào đối tượng `DataPackage`.

Một đối tượng `DataPackage` được thiết kế để đặc tả một nội dung duy nhất. Tuy nhiên, nội dung duy nhất này có thể tồn tại ở nhiều định dạng dữ liệu khác nhau. Ví dụ, một văn bản HTML có thể có định dạng HTML, định dạng Rich Text Format (RTF), và định dạng văn bản thuần túy. Một hình ảnh có thể có định dạng hình ảnh, định dạng văn bản mô tả hình ảnh, hoặc định dạng URI cho biết nguồn gốc của hình ảnh.

Ứng dụng nguồn gọi phương thức `SetData` của `DataPackage` một lần cho mỗi định dạng dữ liệu. Mỗi lần gọi `SetData` sẽ thêm một loại định dạng (một chuỗi String) và dữ liệu (một đối tượng Object) đại diện cho định dạng đó vào một dictionary. WinRT định nghĩa một lớp tĩnh `StandardDataFormats` với các thuộc tính String chỉ đọc mô tả các định dạng chuẩn được định nghĩa sẵn:

```
public static class StandardDataFormats {
   public static String Text               { get; }  // "Text"
   public static String Rtf                { get; }  // "Rich Text Format"
   public static String Html               { get; }  // "HTML Format"
   public static String Bitmap             { get; }  // "Bitmap"
   public static String ApplicationLink    { get; }  // "ApplicationLink"
   public static String WebLink            { get; }  //
"UniformResourceLocatorW"
   public static String StorageItems       { get; }  // "Shell IDList Array"
}
```

Với các định dạng dữ liệu chuẩn, `DataPackage` cung cấp các phương thức đơn giản, mạnh mẽ về kiểu dữ liệu mà bên trong gọi `SetData`, truyền vào một chuỗi từ `StandardDataFormats` và dữ liệu đã được định dạng: `SetText`, `SetRtf`, `SetHtmlFormat`, `SetBitmap`, `SetApplicationLink`, `SetWebLink`, và `SetStorageItems`.

Tuy nhiên, bạn không bị giới hạn trong các định dạng chuẩn này. Một ứng dụng có thể định nghĩa định dạng dữ liệu riêng và thêm định dạng tùy chỉnh này vào đối tượng `DataPackage`. Dĩ nhiên, ứng dụng đích sẽ phải biết cùng tên định dạng để lấy dữ liệu này ra từ đối tượng `DataPackage`. Có nhiều định dạng dữ liệu được công nhận được mô tả tại schema.org dành cho các loại như sách, phim, công thức nấu ăn, sự kiện, con người, địa điểm, nhà hàng, sản phẩm, ưu đãi, đánh giá và nhiều hơn nữa. Nếu bạn muốn sử dụng một trong các định dạng này, hãy xem gói NuGet tại Transhipment trên GitHub. Gói này bao gồm mã để hỗ trợ các định dạng này, giúp bạn giảm thiểu mã cần viết.

Quan trọng: Thứ tự mà một ứng dụng thêm các định dạng vào `DataPackage` rất quan trọng: nó nên thêm các định dạng có độ trung thực cao nhất trước và định dạng có độ trung thực thấp nhất cuối cùng. Ví dụ, văn bản HTML nên được thêm theo thứ tự: HTML, RTF, và sau đó là Text. Ứng dụng đích có thể duyệt qua các định dạng khả dụng theo thứ tự. Nếu một ứng dụng đích không hỗ trợ định dạng HTML, nó có thể sử dụng định dạng RTF (nếu hỗ trợ định dạng này) hoặc định dạng Text (nếu hỗ trợ định dạng này). Tất nhiên, ứng dụng đích có thể không hỗ trợ bất kỳ định dạng nào có trong đối tượng `DataPackage` từ ứng dụng nguồn; trong trường hợp này, ứng dụng đích không thể nhận nội dung.

Chia sẻ qua clipboard

Từ khi ra đời, Windows đã hỗ trợ chia sẻ dữ liệu qua clipboard. Điểm tuyệt vời của clipboard là nó cho phép người dùng chọn chính xác những gì họ muốn chia sẻ. Người dùng cũng quyết định ứng dụng nào sẽ nhận được nội dung được chia sẻ. Miễn là ứng dụng nguồn và ứng dụng đích có ít nhất một định dạng chung, dữ liệu có thể được chia sẻ. Một điểm tuyệt vời khác của clipboard là nó cho phép chia sẻ dữ liệu giữa ứng dụng Windows Store và ứng dụng desktop.

WinRT cung cấp lớp `Windows.ApplicationModel.DataTransfer.Clipboard` để ứng dụng sử dụng và thao tác với clipboard. Vì Windows chỉ có một clipboard duy nhất, lớp này là một lớp tĩnh:

```
// Lớp tĩnh vì chỉ có một clipboard hệ thống
public static class Clipboard {
   // Các phương thức được gọi bởi ứng dụng nguồn chia sẻ:
   public static void Clear(); // Đặt một DataPackage rỗng lên clipboard
   public static void SetContent(DataPackage content); // Thay thế nội dung
hiện có
   public static void Flush();  // Giữ nội dung trên clipboard ngay cả khi
ứng dụng kết thúc

   // Các phương thức được gọi bởi ứng dụng đích chia sẻ:
   public static DataPackageView GetContent(); // Trả về phiên bản chỉ đọc
của DataPackage

   // Được kích hoạt khi nội dung thay đổi (ứng dụng đích sử dụng để biết nếu
thao tác paste khả dụng)
   public static event EventHandler<object> ContentChanged;
}
```

Quan trọng: Để giữ clipboard dưới sự kiểm soát của người dùng, một ứng dụng chỉ nên gọi các phương thức của `Clipboard` từ luồng GUI khi ứng dụng đang ở foreground hoặc đang chạy dưới trình gỡ lỗi. Điều này ngăn các ứng dụng khác hoặc các tác vụ nền xóa hoặc thay đổi nội dung clipboard một cách bất ngờ.

Dưới đây là mã được thực thi bởi ứng dụng nguồn, tạo và khởi tạo một đối tượng `DataPackage` và sau đó đặt nó lên clipboard:

```
private void PutDataPackageOnClipboard(DataPackageOperation operation) {
   DataPackage dp = new DataPackage();
   // Đặt các thuộc tính mong muốn:
   dp.Properties.ApplicationName = Package.Current.DisplayName;
   dp.Properties.Title = "DataPackage Title";
   dp.Properties.Description = "DataPackage Description";
   dp.RequestedOperation = operation;  // None, Copy, Move, hoặc Link

   // Thêm các định dạng dữ liệu mong muốn:
   dp.SetText(m_txt.Text);  // Lấy văn bản từ một TextBox control

   // Tùy chọn: đăng ký các trình xử lý sự kiện:
   dp.OperationCompleted += OnShareCompleted; // Bắt buộc với Move để ứng
dụng nguồn xóa dữ liệu
   dp.Destroyed += OnDataPackageDestroyed;

   // Đặt DataPackage lên clipboard. LƯU Ý: Clipboard tạo một bản sao của
gói;
   // các thay đổi đối với đối tượng DataPackage sẽ không ảnh hưởng đến những
gì đã có trên clipboard.
   Clipboard.SetContent(dp);
}
```

Sau khi ứng dụng nguồn đã đặt nội dung của mình lên clipboard, người dùng có thể chuyển sang một ứng dụng khác, khiến ứng dụng đó trở thành ứng dụng đích. Trong ứng dụng đích, người dùng sẽ kích hoạt thao tác paste bằng một cơ chế nào đó đặc trưng của ứng dụng đích. Nhiều ứng dụng đích hỗ trợ chức năng paste thông qua lệnh thanh công cụ, mục menu, hoặc sử dụng tổ hợp phím Ctrl+V. Đây là những cơ chế tiêu chuẩn mà người dùng đã quen thuộc, nhưng một ứng dụng đích có thể áp dụng bất kỳ cơ chế nào mà nó mong muốn.

Khi được kích hoạt để chấp nhận dữ liệu được chia sẻ qua clipboard, ứng dụng đích gọi phương thức `Clipboard.GetContent` để lấy các đối tượng `DataPackage`. Tuy nhiên, để đảm bảo ứng dụng đích không thể thay đổi nội dung clipboard, một đối tượng `DataPackageView` được trả về; điều này cấp quyền truy cập chỉ đọc tới `DataPackage`, và đối tượng này được định nghĩa như sau:

```
public sealed class DataPackageView {
   // ApplicationName, Title, Description, v.v.
   public DataPackagePropertySetView Properties { get; }
   // Lý do tại sao dữ liệu được chia sẻ (None, Copy, Move, Link)
   public DataPackageOperation RequestedOperation { get; }

   // Sử dụng các phương thức này để kiểm tra các định dạng khả dụng hoặc nếu
một định dạng cụ thể khả dụng:
   public IReadOnlyList<String> AvailableFormats { get; }  // Theo thứ tự từ
độ trung thực cao nhất đến thấp nhất
   public Boolean Contains(String formatId);

   // Gọi GetDataAsync để lấy dữ liệu cho một định dạng được chỉ định
   public IAsyncOperation<Object> GetDataAsync(String formatId);

   // Các phương thức này là wrapper an toàn về kiểu dữ liệu cho GetDataAsync
với các kiểu phổ biến:
   public IAsyncOperation<String> GetTextAsync(String formatId);  // Dành cho
văn bản thường, Rtf, v.v.
   public IAsyncOperation<String> GetTextAsync();                // Dành cho
văn bản thường
   public IAsyncOperation<String> GetRtfAsync();
   public IAsyncOperation<String> GetHtmlFormatAsync();
   public IAsyncOperation<RandomAccessStreamReference> GetBitmapAsync();
   public IAsyncOperation<Uri> GetApplicationLinkAsync();
   public IAsyncOperation<Uri> GetWebLinkAsync();
   public IAsyncOperation<IReadOnlyList<IStorageItem>>
GetStorageItemsAsync();

   // Khi lấy HTML, ứng dụng đích cũng có thể lấy tham chiếu tới nội dung
nguồn bổ sung
   public IAsyncOperation<IReadOnlyDictionary<String,
RandomAccessStreamReference>>
      GetResourceMapAsync();

   // Nói với ứng dụng nguồn ứng dụng đích đã làm gì với nội dung (kích hoạt
sự kiện OperationCompleted)
   public void ReportOperationCompleted(DataPackageOperation value);
}
```

Chia sẻ qua Share charm

Ngoài clipboard, các ứng dụng Windows Store có thể tận dụng một cơ chế khác để chia sẻ dữ liệu giữa các ứng dụng: Share charm. Không giống như clipboard, yêu cầu người dùng chuyển sang ứng dụng đích để chấp nhận dữ liệu được chia sẻ, Share charm được thiết kế để hỗ trợ các tình huống chia sẻ ngữ cảnh nhanh chóng mà người dùng muốn hoàn thành ngay trong ứng dụng nguồn. Share charm cho phép người dùng đăng liên kết video lên mạng xã hội, thêm một liên kết vào ứng dụng danh sách đọc, gửi email URL trang web cho bạn bè, v.v. Nó cũng có thể được sử dụng để gửi một bức ảnh tới ứng dụng chỉnh sửa ảnh, làm cho ứng dụng chỉnh sửa biết đến bức ảnh. Tuy nhiên, người dùng phải tự khởi chạy ứng dụng chỉnh sửa để thao tác với ảnh. Tình huống này sẽ tốt hơn nếu khởi chạy ứng dụng chỉnh sửa thông qua liên kết kiểu tệp hoặc liên kết giao thức (được thảo luận trong phần "Liên kết kiểu tệp" của Chương 5, "Tệp và thư mục lưu trữ").

Đối với các nhà phát triển, cơ chế Share charm cho phép họ tập trung vào việc làm tốt chức năng của ứng dụng. Ứng dụng nguồn không cần biết cách đăng nội dung lên mạng xã hội, cách duy trì danh sách đọc, hoặc cách gửi email. Các ứng dụng đích khác nhau biết cách làm những điều này, và người dùng khởi tạo hành động thông qua Share charm.

Quy trình cơ bản của Share charm như sau:

1. Ứng dụng nguồn có một nội dung mà người dùng muốn chia sẻ.
2. Người dùng chọn Share charm bằng cách vuốt từ mép màn hình và chọn charm, sử dụng chuột để chọn Share charm, hoặc nhấn Windows+H.
3. Ứng dụng nguồn được thông báo rằng người dùng đã chọn Share charm. Tại thời điểm này, ứng dụng nguồn tạo và khởi tạo một đối tượng `DataPackage`, điền nội dung mà ứng dụng muốn chia sẻ. Ứng dụng nguồn nên chèn tất cả các định dạng dữ liệu áp dụng. Ứng dụng này đưa đối tượng `DataPackage` cho Windows.
4. Windows quét các định dạng dữ liệu được cung cấp trong đối tượng `DataPackage` và sau đó kiểm tra tất cả các ứng dụng Windows Store đã được cài đặt mà đã khai báo hỗ trợ kích hoạt chia sẻ dữ liệu (đôi khi được gọi là hợp đồng chia sẻ) trong tệp manifest của ứng dụng. Khi một ứng dụng khai báo hỗ trợ kích hoạt chia sẻ dữ liệu, nó cũng chỉ định các định dạng dữ liệu mà ứng dụng có thể hiểu. Dựa trên các định dạng dữ liệu có trong đối tượng `DataPackage`, Windows xác định ứng dụng nào đã cài đặt có thể chấp nhận các định dạng này và hiển thị danh sách các ứng dụng đích cho người dùng trong Share pane, như được hiển thị trong Hình 10-1.

Hình 10-1: Chia sẻ từ ứng dụng Weather hiển thị Share pane, liệt kê các ứng dụng đích hỗ trợ các định dạng mà ứng dụng Weather đã đặt trong `DataPackage`.

5. Lúc này, người dùng có thể chọn một ứng dụng đích, khiến Windows kích hoạt ứng dụng đó (thông qua kích hoạt dạng hosted-view). Windows chuyển đối tượng `DataPackageView` cho ứng dụng đích, cấp quyền truy cập chỉ đọc vào `DataPackage`. Ứng dụng đích sẽ hiển thị giao diện người dùng, cho biết ứng dụng có thể truy cập dữ liệu được chia sẻ và ý định của nó với dữ liệu này (đăng, thêm vào một bộ sưu tập, gửi email, v.v.). Ứng dụng đích cũng nên cho người dùng biết cần xác nhận hành động này;

nó không nên thực hiện hành động mà không có sự đồng ý của người dùng. Hình 10-2 minh họa giao diện ứng dụng Mail, nơi người dùng có thể quyết định gửi dữ liệu đến ai hoặc nhấn nút Back để hủy thao tác chia sẻ.

Hình 10-2: Giao diện ứng dụng Mail hiển thị ý định với nội dung chia sẻ. Người dùng có thể chấp nhận hoặc hủy bỏ.

6. Sau khi người dùng xác nhận hoặc từ chối hành động được đề xuất bởi ứng dụng đích, giao diện hosted-view của ứng dụng đích bị ẩn và người dùng được trả về ứng dụng nguồn để tiếp tục tương tác. Cửa sổ hosted-view bị hủy sau khi hoàn thành thao tác chia sẻ.

Lưu ý: Cơ chế Share charm được thiết kế chủ yếu để chia sẻ giữa một ứng dụng Windows Store với một ứng dụng Windows Store khác; nó không được thiết kế để chia sẻ với các ứng dụng desktop. Tuy nhiên, khi sử dụng một ứng dụng desktop, nếu người dùng chọn Share charm, Windows tự động tạo một `DataPackage` chứa ảnh chụp màn hình của toàn bộ desktop và cho phép hình ảnh bitmap được chia sẻ với các ứng dụng Windows Store hỗ trợ hình ảnh bitmap. Đối với các ứng dụng Windows Store, hệ thống cũng tạo một `DataPackage` chứa ảnh chụp màn hình của cửa sổ active-view và làm cho nó có sẵn để chia sẻ. Ngoài ra, với các ứng dụng Windows Store được cài đặt qua Windows Store, hệ thống cũng tạo một `DataPackage` chứa URI đến ứng dụng trên Windows Store. URI này được định dạng dưới dạng "Text", "UniformResourceLocatorW", và "HTML Format" để bất kỳ ứng dụng nào hỗ trợ ba định dạng này có thể chấp nhận URI được chia sẻ.

Implementing a share source app

Dự kiến rằng hầu hết các ứng dụng Windows Store sẽ là ứng dụng nguồn chia sẻ. Lý do là vì hầu hết các ứng dụng đều có loại nội dung nào đó có thể chia sẻ được. Nếu người dùng chọn một nội dung cụ thể trong ứng dụng, thì nội dung này nên được chia sẻ khi người dùng chọn Share charm. Nếu người dùng không chọn nội dung cụ thể nào, ứng dụng nên chia sẻ ngầm định một thứ gì đó có ý nghĩa dựa trên những gì đang được hiển thị cho người dùng. Ví dụ, khi chia sẻ từ Internet Explorer, URI của trang web hiện tại sẽ được chia sẻ trừ khi người dùng đã chọn một đoạn văn bản cụ thể trên trang. Trong trường hợp này, đoạn văn bản được chia sẻ.

Để nhận thông báo khi người dùng chọn Share charm, một ứng dụng nguồn chia sẻ phải sử dụng lớp `DataTransferManager` của WinRT:

```
public sealed class DataTransferManager {
   // Lấy DataTransferManager liên kết với view chính hoặc phụ của ứng dụng
(không phải hosted view)
   public static DataTransferManager GetForCurrentView();
```

```
   // Được kích hoạt khi người dùng chọn Share charm khi view đang được kích
hoạt
   public event TypedEventHandler<DataTransferManager,
DataRequestedEventArgs> DataRequested;

   // Được kích hoạt khi người dùng chọn một ứng dụng đích (hiếm khi dùng),
   // để ứng dụng nguồn biết tên của ứng dụng đích
   public event TypedEventHandler<DataTransferManager,
TargetApplicationChosenEventArgs>
      TargetApplicationChosen;

   // Gọi phương thức này để kích hoạt Share charm theo chương trình
   public static void ShowShareUI();
}
```

Trước tiên, ứng dụng phải lấy tham chiếu đến đối tượng `DataTransferManager` liên kết với view chính hoặc một trong các view phụ của ứng dụng thông qua phương thức tĩnh `GetForCurrentView`. Hosted views không thể sử dụng đối tượng `DataTransferManager`. Sau đó, ứng dụng cần đăng ký một phương thức callback với sự kiện `DataRequested` của `DataTransferManager`:

```
DataTransferManager dtm = DataTransferManager.GetForCurrentView();
dtm.DataRequested += OnDataRequested;
```

Sự kiện này khá đặc biệt vì nó không tích lũy. Tức là, nó chỉ gọi phương thức callback được đăng ký gần đây nhất. Điều này rất tiện lợi, cho phép ứng dụng đăng ký lại cùng một callback mỗi khi ứng dụng có một kích hoạt view chính mà không phải gọi lại cùng một phương thức nhiều lần. Hoặc, một số ứng dụng có thể đăng ký callback với sự kiện này khi phương thức ảo `OnWindowCreated` của ứng dụng được gọi, vì phương thức này đảm bảo được gọi chỉ một lần cho mỗi view chính hoặc phụ.

Khi người dùng chọn Share charm trong ứng dụng, sự kiện `DataRequested` được kích hoạt, gọi phương thức callback. Đối số `DataRequestedEventArgs` cung cấp một thuộc tính `Request`, trả về đối tượng `DataRequest`:

```
public sealed class DataRequest {
   public DataPackage Data { get; set; }  // Trả về một DataPackage để khởi
tạo
   public void FailWithDisplayText(String value); // Gọi nếu ứng dụng không
thể chia sẻ nội dung ngay bây giờ

   public DataRequestDeferral GetDeferral(); // Gọi để thực hiện các thao tác
bất đồng bộ trong handler
   public DateTimeOffset Deadline { get; }  // Chỉ ra thời hạn cho việc kết
xuất bị trì hoãn
}
```

Bạn thường triển khai phương thức callback như sau:

```
private void OnDataRequested(DataTransferManager dtm, DataRequestedEventArgs
e) {
```

```
   if (s_nothingToShareRightNow) {  // Thay thế điều kiện này bằng logic của
bạn
      e.Request.FailWithDisplayText("Select something to share.");
      return;
   }

   // 1. Đặt các thuộc tính của DataPackage:
   e.Request.Data.Properties.ApplicationName = Package.Current.DisplayName;
   e.Request.Data.Properties.Title = "DataPackage Title";
   e.Request.Data.Properties.Description = "DataPackage Description";
   RandomAccessStreamReference image =
      RandomAccessStreamReference.CreateFromUri(new Uri("ms-
appx:///Assets/Planets.png"));
   e.Request.Data.Properties.Thumbnail = image;

   // 2. Thêm các định dạng dữ liệu mong muốn vào DataPackage:
   e.Request.Data.SetText("Some text");
   e.Request.Data.SetBitmap(image);
}
```

Delayed rendering of shared content

Windows khuyến nghị rằng các ứng dụng nên điền dữ liệu vào `DataPackage` trong vòng 200 mili-giây hoặc ít hơn. Tuy nhiên, việc lấy một số dữ liệu có thể tốn nhiều thời gian (ví dụ: nếu dữ liệu cần được tải xuống, nén, hoặc mã hóa). Ngoài ra, sẽ không tốt nếu cấp phát bộ nhớ lớn cho dữ liệu mà ứng dụng đích không sử dụng. Để giải quyết hai vấn đề này, Windows hỗ trợ kết xuất bị trì hoãn cho định dạng dữ liệu của `DataPackage`.

Nếu việc lấy dữ liệu để đưa vào `DataPackage` mất nhiều thời gian hoặc tiêu tốn nhiều bộ nhớ, bạn có thể sử dụng phương thức `SetDataProvider` của `DataPackage`. Phương thức này cho phép bạn chỉ định một định dạng dữ liệu và một delegate trỏ đến một phương thức callback. Tuy nhiên, Windows chỉ gọi phương thức callback khi dữ liệu liên quan đến định dạng đó được yêu cầu. Điều này cho phép dữ liệu được lấy theo nhu cầu khi ứng dụng đích thực sự muốn dữ liệu. Bạn có thể gọi `SetDataProvider` nhiều lần, mỗi lần cho một định dạng dữ liệu khác nhau.

Ví dụ về cách triển khai phương thức callback:

```
private async void DataProviderHandler(DataProviderRequest request) {
   DataProviderDeferral deferral = request.GetDeferral();
   try {
      switch (request.FormatId) {
         case "Bitmap":
            // Thực hiện thao tác tiêu tốn tài nguyên hoặc chờ I/O tại đây
            String data = await GetBitmapAsync();
            request.SetData(data);
            break;
      }
   }
   finally {
      deferral.Complete();
   }
```

```
}
```

Khi sử dụng kỹ thuật này với các đối tượng `StorageItem`, bạn phải thêm các kiểu tệp vào thuộc tính của `DataPackage` trước để hệ thống biết các kiểu tệp nào có trong `DataPackage`. Điều này cho phép hệ thống nhanh chóng xác định các ứng dụng đích hỗ trợ các kiểu tệp đã chỉ định:

```
DataPackage dp = ...;
dp.Properties.FileTypes.Add(".jpg");
dp.Properties.FileTypes.Add(".png");
dp.SetDataProvider(StandardDataFormats.StorageItems, DataProviderHandler);
```

Implementing a share target app

Mặc dù nhiều ứng dụng có thể và nên là ứng dụng nguồn chia sẻ, rất ít ứng dụng nên là ứng dụng đích chia sẻ, vì nhiều ứng dụng không có mục đích xử lý nội dung được chia sẻ. Tuy nhiên, một số ứng dụng rất phù hợp để trở thành ứng dụng đích chia sẻ, như các ứng dụng giao tiếp (mạng xã hội và email), ứng dụng ghi chú, và một số ứng dụng có thể truyền dữ liệu tới các thiết bị.

Việc tạo một ứng dụng đích chia sẻ khá đơn giản. Đầu tiên, bạn phải thêm khai báo **Share Target** vào tệp manifest của ứng dụng. Khi thêm khai báo này, bạn cần chỉ định thông qua trường **Share Description** nội dung ứng dụng sẽ làm với dữ liệu được chia sẻ. Văn bản này sẽ xuất hiện trong Share pane và giúp người dùng hiểu ứng dụng đích sẽ làm gì với dữ liệu mà không cần kích hoạt ứng dụng. Ví dụ, ứng dụng Reading List hiển thị "Bookmark for later" trong Share pane.

Trong manifest, bạn cũng cần chỉ định các định dạng dữ liệu mà ứng dụng đích chia sẻ của bạn hỗ trợ. Ví dụ, "Text", "Rtf", "HTML Format", và "Bitmap". Hình 10-3 minh họa cách thêm khai báo **Share Target**. Việc chỉ định bất kỳ kiểu tệp nào trong phần **Supported File Types** đồng nghĩa với việc hỗ trợ định dạng dữ liệu "StorageItems". Nếu bạn chọn **Supports Any File Type**, ứng dụng đích của bạn sẽ xuất hiện trong Share charm nếu có bất kỳ phần tử lưu trữ nào trong `DataPackage`. Nếu bạn chỉ định một số kiểu tệp cụ thể và không chọn **Supports Any File Type**, ứng dụng sẽ chỉ xuất hiện nếu kiểu tệp trong `DataPackage` khớp với kiểu tệp mà ứng dụng hỗ trợ.

Tiếp theo, bạn cần xác định một lớp kế thừa từ `Page` trong XAML, lớp này hiển thị giao diện người dùng cho phép người dùng xác nhận hoặc hủy bỏ quá trình chia sẻ với ứng dụng của bạn. Nếu muốn, bạn có thể để Microsoft Visual Studio khởi tạo nhanh lớp này bằng cách chọn **Project > Add New Item > Visual C# > Windows Store > Share Target Contract**. Thao tác này sẽ tạo một lớp cơ bản kế thừa từ `Page` với một số mã backend giúp hiển thị các thuộc tính `Title`, `Description`, và `Thumbnail` của `DataPackage` trong giao diện XAML.

Khi `DataPackageView` được chuyển tới ứng dụng đích, hệ thống sẽ tự động điền trước các thuộc tính `LogoBackgroundColor` và `Square30x30Logo`. (Một ứng dụng nguồn chia sẻ có thể thay đổi các giá trị này nếu muốn.) Những giá trị này có thể được hiển thị bởi ứng dụng đích để người dùng biết ứng dụng nguồn nào đã cung cấp dữ liệu được chia sẻ. Ví dụ, ứng dụng Reading List hiển thị thông tin nguồn của ứng dụng với từng mục.

Để hiển thị trang này, bạn cần ghi đè phương thức `OnShareTargetActivated` của lớp `App`. Khi người dùng chọn ứng dụng của bạn từ Share pane, Windows kích hoạt ứng dụng của bạn ở chế độ hosted view và gọi phương thức `OnShareTargetActivated`, chuyển một đối tượng `ShareTargetActivatedEventArgs` làm tham số. Đối tượng này có thuộc tính `ShareOperation`, trả về một đối tượng `ShareOperation`:

```
public sealed class ShareOperation {
   // 1. Ứng dụng đích lấy DataPackageView để hiển thị dữ liệu được chia sẻ
   public DataPackageView Data { get; }

   // 2. Ứng dụng đích gọi phương thức này sau khi người dùng đồng ý chia sẻ
dữ liệu
   public void ReportStarted(); // Giữ ứng dụng nguồn hoạt động, thêm thao
tác chia sẻ vào danh sách tiến trình

   // 3. Ứng dụng đích gọi phương thức này để ẩn hosted view, cho phép người
dùng quay lại ứng dụng nguồn
   public void DismissUI();

   // 4. Ứng dụng đích gọi một trong các phương thức sau sau khi hoàn thành
thao tác chia sẻ
   public void ReportCompleted(); // Loại bỏ thao tác chia sẻ khỏi danh sách
tiến trình
   public void ReportError(String value); // Hiển thị thông báo lỗi và cập
nhật danh sách tiến trình

   // Các phương thức cho thao tác chia sẻ kéo dài:
   public void ReportDataRetrieved(); // Cho phép ứng dụng nguồn bị treo hoặc
kết thúc
   public void ReportSubmittedBackgroundTask(); // Cho phép ứng dụng đích bị
treo

   // Các phương thức dùng để thao tác với quick links:
   public void ReportCompleted(QuickLink quicklink); // Thêm quick link khi
hoàn tất chia sẻ
   public String QuickLinkId { get; } // Trả về Id của QuickLink
   public void RemoveThisQuickLink();
}
```

Trong phương thức `OnShareTargetActivated`, bạn cần tạo một trang giao diện XAML, lưu tham chiếu tới đối tượng `ShareOperation`, và kích hoạt trang. Mã kích hoạt cần thực thi nhanh chóng, nếu không Windows sẽ nghĩ rằng hosted view không phản hồi và sẽ đóng toàn bộ tiến trình (bao gồm cả view chính nếu đang chạy). Sau khi hosted view được kích hoạt, mã của bạn có thể truy cập thuộc tính `Data` của đối tượng `ShareOperation`. Thuộc tính này trả về một đối tượng `DataPackageView` chứa dữ liệu đã được chuyển tới ứng dụng đích. Tại thời điểm này, mã có thể trích xuất một số thuộc tính và định dạng dữ liệu để điền vào giao diện người dùng của hosted view như mong muốn.

Cuối cùng, trang đích chia sẻ của ứng dụng bạn cần hiển thị cho người dùng biết dữ liệu nào sẽ được xử lý và ứng dụng dự định làm gì với dữ liệu đó. Bạn có thể cho phép người dùng tùy chỉnh dữ liệu, như thêm văn bản vào nội dung email, chỉnh sửa tiêu đề, hoặc chỉ định định dạng

dữ liệu mà ứng dụng đích sẽ xử lý. Khi người dùng đã hài lòng, họ sẽ nhấn nút **Share** hoặc **Send** trong giao diện trang đích để kích hoạt thao tác chia sẻ.

Implementing an extended (lengthy) share operation

Đôi khi, việc chia sẻ dữ liệu có thể mất nhiều thời gian, đặc biệt nếu ứng dụng đích đang thực hiện việc truyền tải tệp. Những tình huống này có thể gây ra vấn đề vì các quy tắc **Windows Process Lifetime Management (PLM)**. Ví dụ, hệ thống sẽ treo các luồng trong ứng dụng đích nếu người dùng không tương tác với nó, ngăn ứng dụng hoàn thành thao tác chia sẻ. Do đó, khi thực hiện một thao tác chia sẻ kéo dài, ứng dụng đích cần gọi một số phương thức bổ sung.

Cụ thể, sau khi ứng dụng đích đã hoàn thành việc sử dụng đối tượng `DataPackageView`, ứng dụng nên gọi phương thức `ReportDataRetrieved` của `ShareOperation`. Phương thức này báo cho Windows rằng ứng dụng đích không còn cần ứng dụng nguồn chạy nữa, cho phép hệ thống treo hoặc kết thúc ứng dụng nguồn.

Nếu ứng dụng đích sử dụng các API truyền tải nền (được thảo luận trong Chương 7, “Networking”) để truyền tệp qua mạng, nó nên gọi phương thức `ReportSubmittedBackgroundTask` của `ShareOperation`. Phương thức này báo cho Windows rằng ứng dụng đích có thể bị treo vì thao tác truyền tải hiện đang được thực hiện bởi trình quản lý truyền tải nền.

Share target app quick links

Một số ứng dụng có các "đích đến" cụ thể trong ứng dụng. Ví dụ, ứng dụng Mail cho phép bạn gửi đến một liên hệ cụ thể, và một ứng dụng mạng xã hội có thể cho phép bạn đăng lên trang cá nhân hoặc nhóm cụ thể. Người dùng thường chia sẻ tới cùng một đích đến lặp đi lặp lại. Sẽ rất phiền nếu người dùng phải chọn ứng dụng đích và sau đó chỉ định đích đến nhiều lần. Sẽ tốt hơn nếu người dùng có cách chia sẻ nhanh tới đích đến của một ứng dụng. May mắn thay, Windows hỗ trợ điều này thông qua cơ chế **quick links**.

Quick links xuất hiện ở đầu Share pane.

Sau khi người dùng thành công chia sẻ dữ liệu với đích đến của ứng dụng đích, ứng dụng đích có thể tạo và khởi tạo một đối tượng `QuickLink`:

```
public sealed class QuickLink {
   public QuickLink();

   // Thuộc tính bắt buộc:
   public String Title { get; set; }                        // Hiển thị trong
Share pane
   public RandomAccessStreamReference Thumbnail { get; set; } // Hiển thị
trong Share pane
   public String Id { get; set; }   // Được ứng dụng xác định: chuyển đến ứng
dụng đích thông qua thuộc tính QuickLinkId của ShareOperation
```

```
   // Ít nhất một trong các thuộc tính sau phải được đặt để chỉ định các định
dạng dữ liệu
   // và/hoặc kiểu tệp mà Share pane sử dụng để hiển thị quick link
   public IList<String> SupportedDataFormats { get; }
   public IList<String> SupportedFileTypes { get; }
}
```

Sau khi khởi tạo, đối tượng `QuickLink` có thể được chuyển tới phương thức `ReportCompleted` của `ShareOperation`.

Bây giờ, khi người dùng kích hoạt Share charm, nếu bất kỳ định dạng dữ liệu hoặc kiểu tệp nào trong `DataPackage` khớp với các định dạng dữ liệu hoặc kiểu tệp trong một QuickLink, hệ thống sẽ hiển thị thumbnail và tiêu đề của QuickLink. Nếu người dùng chọn QuickLink, hệ thống sẽ kích hoạt ứng dụng đích và trong phương thức `OnShareTargetActivated` của ứng dụng, nó có thể kiểm tra thuộc tính `QuickLinkId` của `ShareOperation`. Thuộc tính này trả về giá trị được đặt trong thuộc tính `Id` của đối tượng `QuickLink`. Ứng dụng có thể sử dụng giá trị này để nhanh chóng chia sẻ tới một đích đến của nó.

Nếu `Id` của QuickLink không còn hợp lệ, ứng dụng đích có thể gọi phương thức `RemoveThisQuickLink` của `ShareOperation` để xóa nó, đảm bảo nó không còn xuất hiện trong Share pane.

Debugging share target apps

Việc gỡ lỗi ứng dụng đích chia sẻ có thể khá thách thức. Bạn có thể đặt breakpoint trong mã kích hoạt, nhưng khi trình gỡ lỗi dừng ở breakpoint, hành vi light dismiss của Windows ngay lập tức hủy thao tác chia sẻ.

Cách dễ nhất để gỡ lỗi ứng dụng đích chia sẻ là chạy nó trong trình giả lập hoặc trên một PC từ xa. Ngoài ra, bạn cần khởi chạy ứng dụng theo cách thông thường trước, cho phép bạn đặt các breakpoint trong mã kích hoạt của ứng dụng đích. Hoặc, bạn có thể mở **Project Properties > Debug > Do Not Launch, But Debug My Code When It Starts** để thiết lập cấu hình.