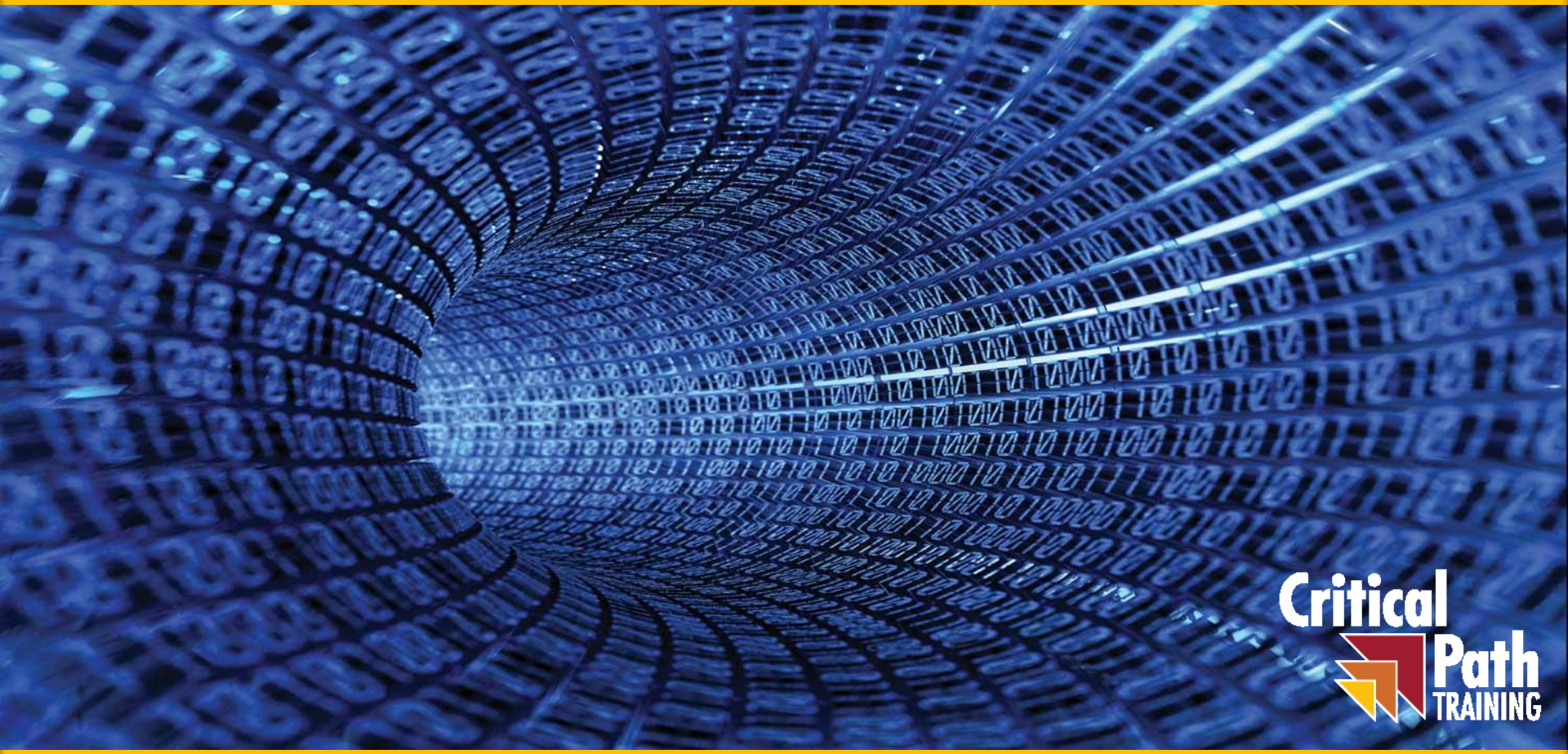
# SharePoint Lists and Events



# Agenda

- Site Columns and Content Types
- Creating Lists and Document Libraries
- Updating Columns, Content Types and Lists
- Creating Event Receivers
- Creating Remote Event Receivers

# Site Columns

- Reusable column templates that define…
  - The underlying field type for column value
  - The default value
  - Rendering characteristics
- Each site has its own Site Column Gallery
  - Site columns available in current site and sites below
  - Site columns in top site available to site collection

Site Settings › Site Columns

Create

| Site Column | Type | Source |
|---|---|---|
| **Wingtip** | | |
| List Price | Currency | Wingtip Intranet |
| Maximum Age | Number | Wingtip Intranet |
| Minimum Age | Number | Wingtip Intranet |
| Product Code | Single line of text | Wingtip Intranet |
| Product Color | Choice | Wingtip Intranet |
| Product Description | Multiple lines of text | Wingtip Intranet |
| Product Image Url | Hyperlink or Picture | Wingtip Intranet |

# Creating Site Columns using XML

- Site columns can be created declaratively
  - Declarative XML element activated using feature

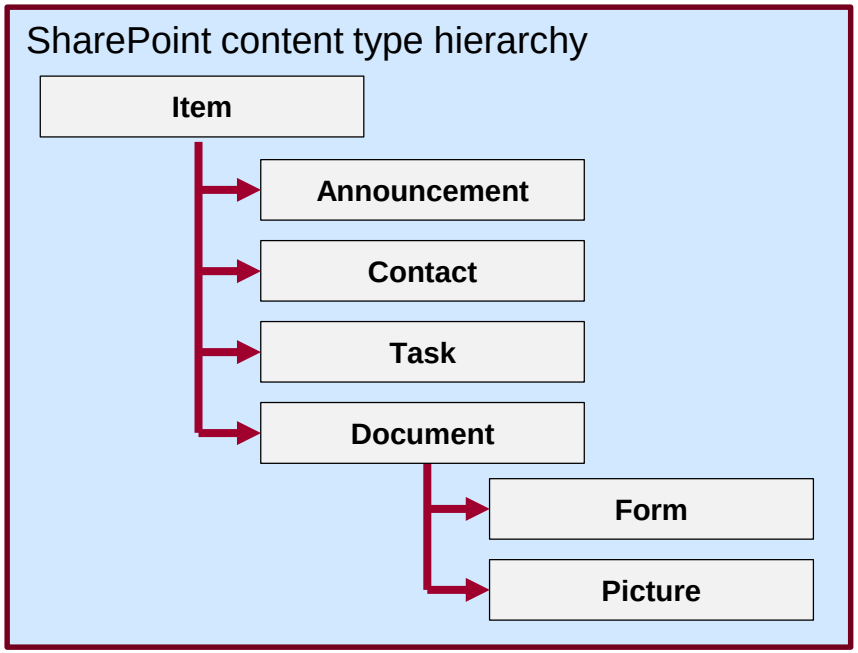
```
<Field
  ID="{11e6b032-2d81-4068-9766-75bb26271e31}"
  Name="BookAuthor"
  DisplayName="Author"
  Type="Text"
  Required="TRUE"
  Group="Wingtip Site Columns" />

<Field
  ID="{7bb22fe4-ca40-4e15-818d-74eb401be8c3}"
  Name="YearPublished"
  DisplayName="Year Published"
  Type="Text"
  Required="TRUE"
  Group="Wingtip Site Columns" />

<Field
  ID="{732082d9-3288-4ce8-92bc-2ba8bf4f39e2}"
  Name="AuthorCountry"
  DisplayName="Author Country"
  Type="Text"
  Required="TRUE"
  Group="Wingtip Site Columns" />

<Field
  ID="{f5b18ca4-d41c-46e5-a4a1-f1703ede46a1}"
  Name="OriginalLanguage"
  DisplayName="Original Language"
  Type="Text"
  Required="TRUE"
  Group="Wingtip Site Columns" />
```

# Content Types

- Reusable item/document templates that define…
  - A parent content type
  - A collection of site columns
- Each site has its own Content Type Gallery
  - Content types available in current site and sites below
  - Content types in top site available to site collection

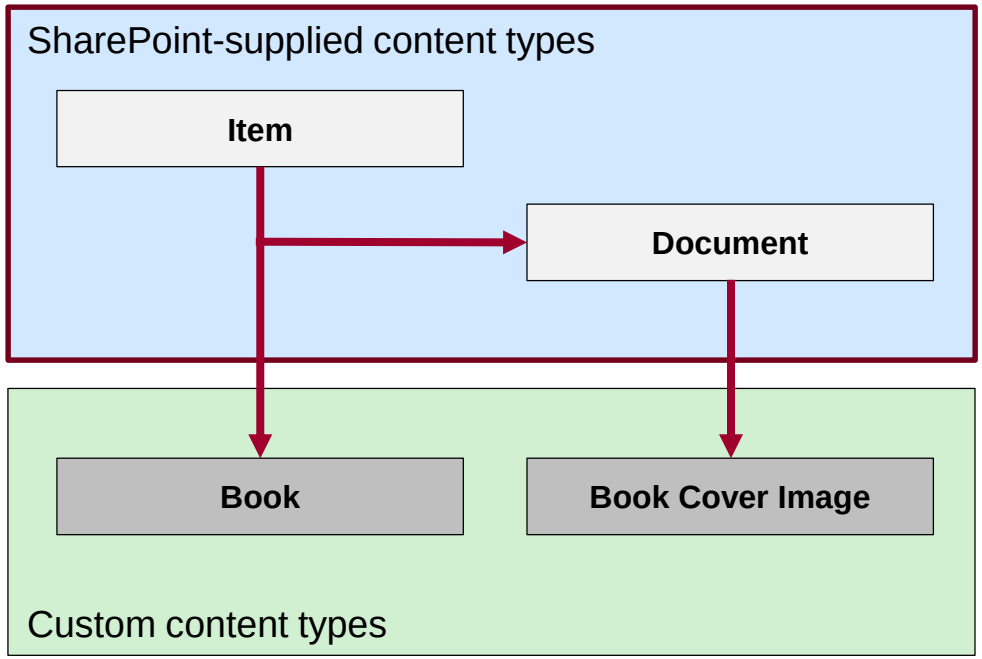
# Content Type Hierarchy

- Content types designed in hierarchy
  - All content types inherit (aka derive) from **Item**
  - Child content type inherits site columns from parent
  - Child content type can add new site columns
  - Child content type can remove site columns inherited from parent



# Creating Custom Content Types

- Creating a custom content type
  - Select a content type name
  - Select a parent content type to inherit from
  - Add whatever site columns are required
  - Configure content type settings



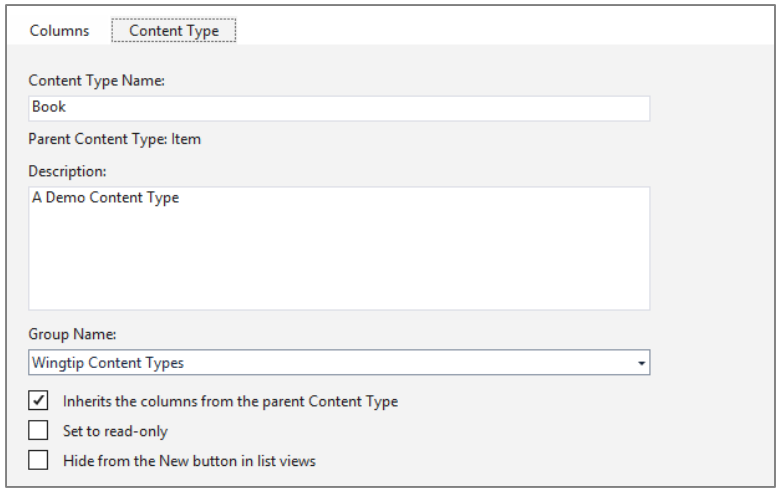
# Creating Content Types using XML

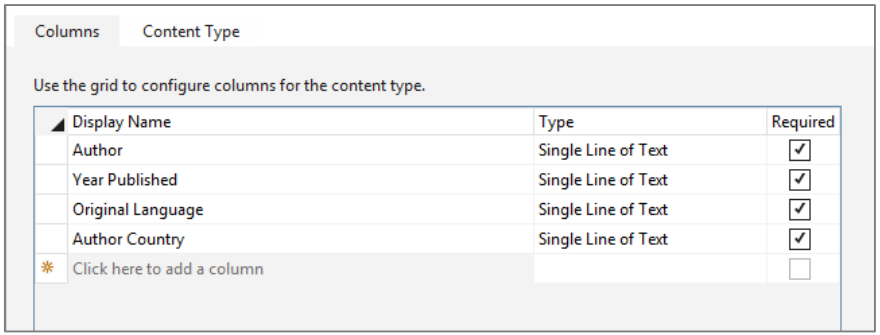
- This example shows a custom content type
  - Designed for use custom Books list type

```xml
<!-- Parent ContentType: Item (0x01) -->
<ContentType
  ID="0x0100F5EB0315D6B0413C9503E2E54A52363F"
  Name="Book"
  Group="Wingtip Content Types"
  Description="A Demo Content Type"
  Inherits="TRUE"
  Version="0">

  <FieldRefs>
    <FieldRef ID="{11e6b032-2d81-4068-9766-75bb26271e31}" Name="BookAuthor" />
    <FieldRef ID="{7bb22fe4-ca40-4e15-818d-74eb401be8c3}" Name="YearPublished" />
    <FieldRef ID="{f5b18ca4-d41c-46e5-a4a1-f1703ede46a1}" Name="OriginalLanguage" />
    <FieldRef ID="{732082d9-3288-4ce8-92bc-2ba8bf4f39e2}" Name="AuthorCountry" />
  </FieldRefs>

</ContentType>
```

# Visual Studio Content Type Designer

- Makes it easier to work with content types
  - You don't need to work directly with XML elements
  - Site columns added to content type from dropdown list

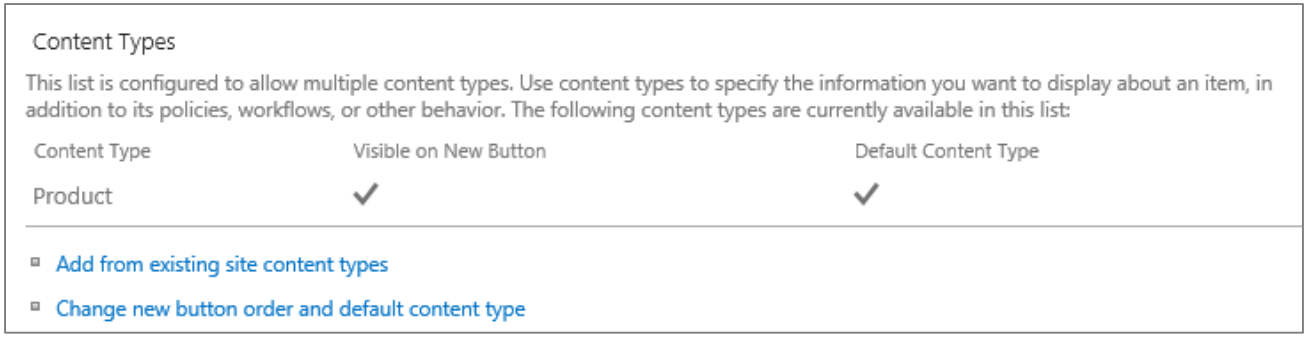


# Lists and Content Types

- List contains a collection of content types
  - Every list must contain at least one content type
  - Content types hidden on the List Settings page by default
  - Advanced Settings page for list provides option  to show them


  - Content Types section allows for adding/removing content types

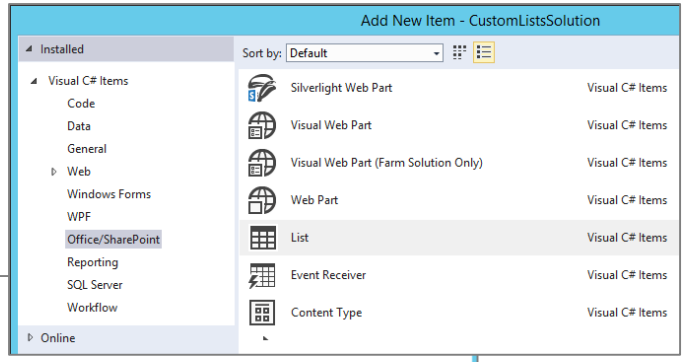


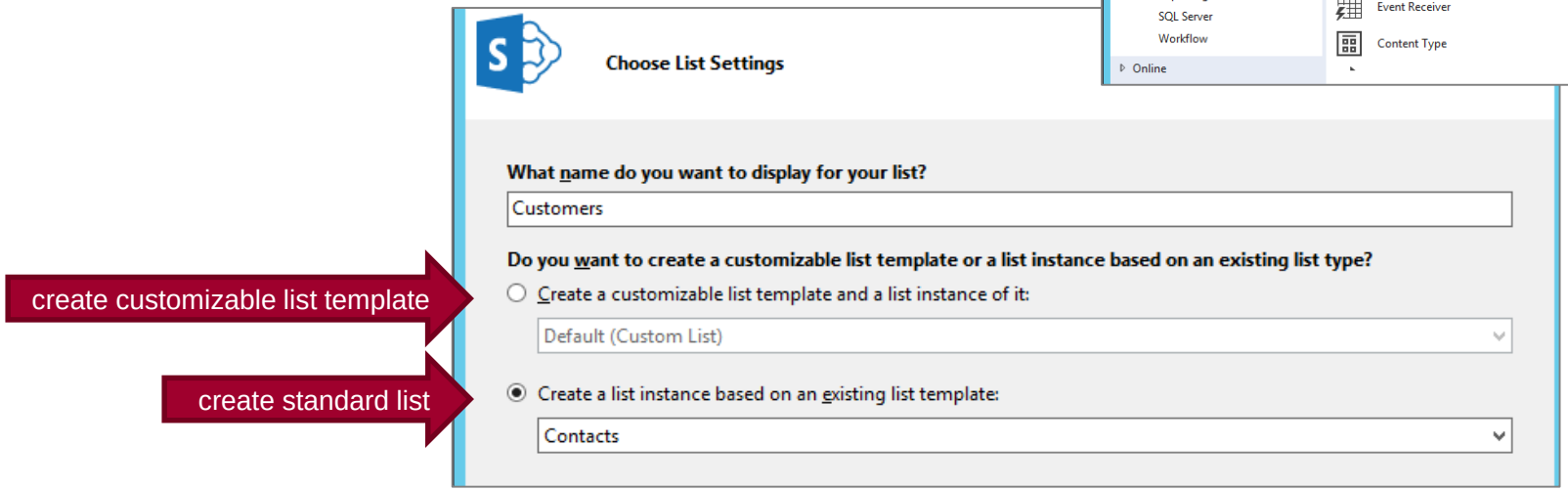
# Agenda

- ✓ Site Columns and Content Types
- ➢ Creating Lists and Document Libraries
- Updating Columns, Content Types and Lists
- Creating Event Receivers
- Creating Remote Event Receivers

# Creating Lists with Visual Studio

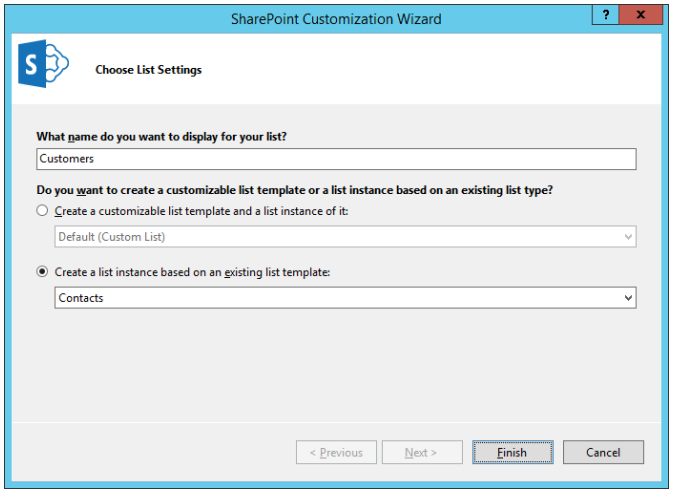
- Visual Studio provides support for creating SharePoint lists
  - SharePoint list added to Visual Studio projects using **List** item template
  - SharePoint lists supported in SharePoint solutions and SharePoint apps
- Options when creating list
  - list instance based on existing list type
  - customizable list template with instance

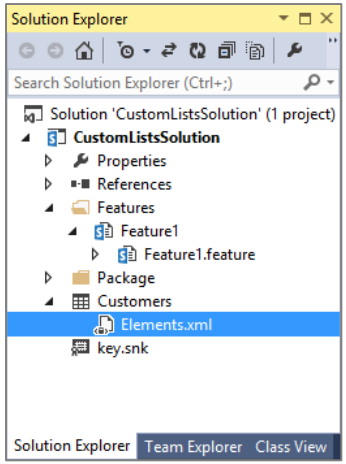


# Creating a Standard List Instance

- Creating using **ListInstance** element
  - **Title**: acts as list display name
  - **URL**: URL offset from root of site
  - **TemplateType**: ID of standard SharePoint list type
  - **FeatureId**: ID of SharePoint feature which provide list type




```
<ListInstance
  Title="Customers"
  OnQuickLaunch="TRUE"
  TemplateType="105"
  FeatureId="00bfea71-7e6d-4186-9ba8-c047ac750105"
  Url="Lists/Customers"
  Description="My List Instance">

</ListInstance>
```

# Populating a New List with Items

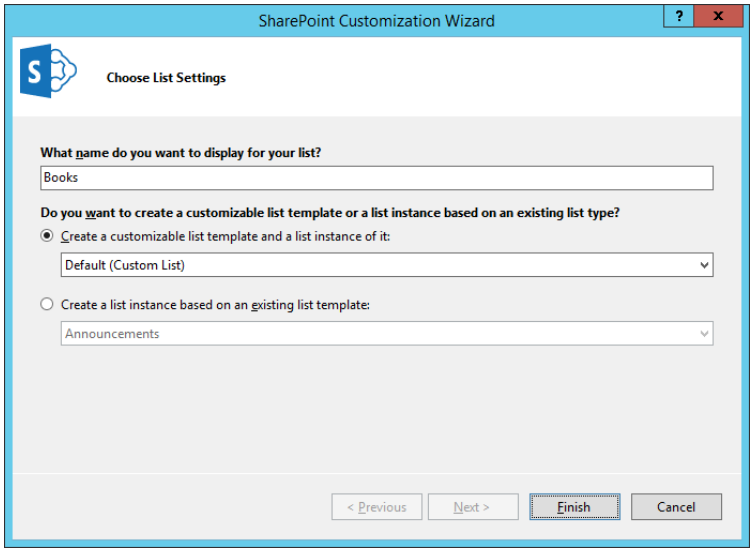
- List can be created with pre-populated items
  - Item data added in Rows and Row elements
  - Item column values assigned using site column names

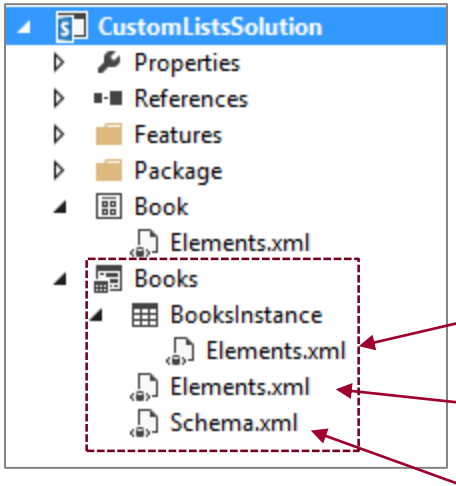
```
<ListInstance
  Title="Customers"
  OnQuickLaunch="TRUE"
  TemplateType="105"
  FeatureId="00bfea71-7e6d-4186-9ba8-c047ac750105"
  Url="Lists/Customers"
  Description="My List Instance">

  <Data>
    <Rows>...</Rows>
  </Data>

</ListInstance>
```

```
<Data>
  <Rows>
    <Row>
      <Field Name="FirstName">Quincy</Field>
      <Field Name="Title">Nelson</Field>
      <Field Name="Company">Benthic Petroleum</Field>
      <Field Name="WorkPhone">1(340)608-7748</Field>
      <Field Name="HomePhone">1(340)517-3737</Field>
      <Field Name="Email">Quincy.Nelson@BenthicPetroleum.com</Field>
    </Row>
    <Row>
      <Field Name="FirstName">Jude</Field>
      <Field Name="Title">Mason</Field>
      <Field Name="Company">Cyberdyne Systems</Field>
      <Field Name="WorkPhone">1(203)408-0466</Field>
      <Field Name="HomePhone">1(203)411-0071</Field>
      <Field Name="Email">Jude.Mason@CyberdyneSystems.com</Field>
    </Row>
    <Row>
      <Field Name="FirstName">Sid</Field>
      <Field Name="Title">Stout</Field>
      <Field Name="Company">Roxxon</Field>
      <Field Name="WorkPhone">1(518)258-6571</Field>
```

# Creating Customizable List Templates

- Select standard List type when creating template
  - Affects initial set of list columns and list content type
  - Schema.xml contains SharePoint list definition
  - First Elements.xml file added with ListTemplate element
  - Second Elements.xml file added with ListInstance element



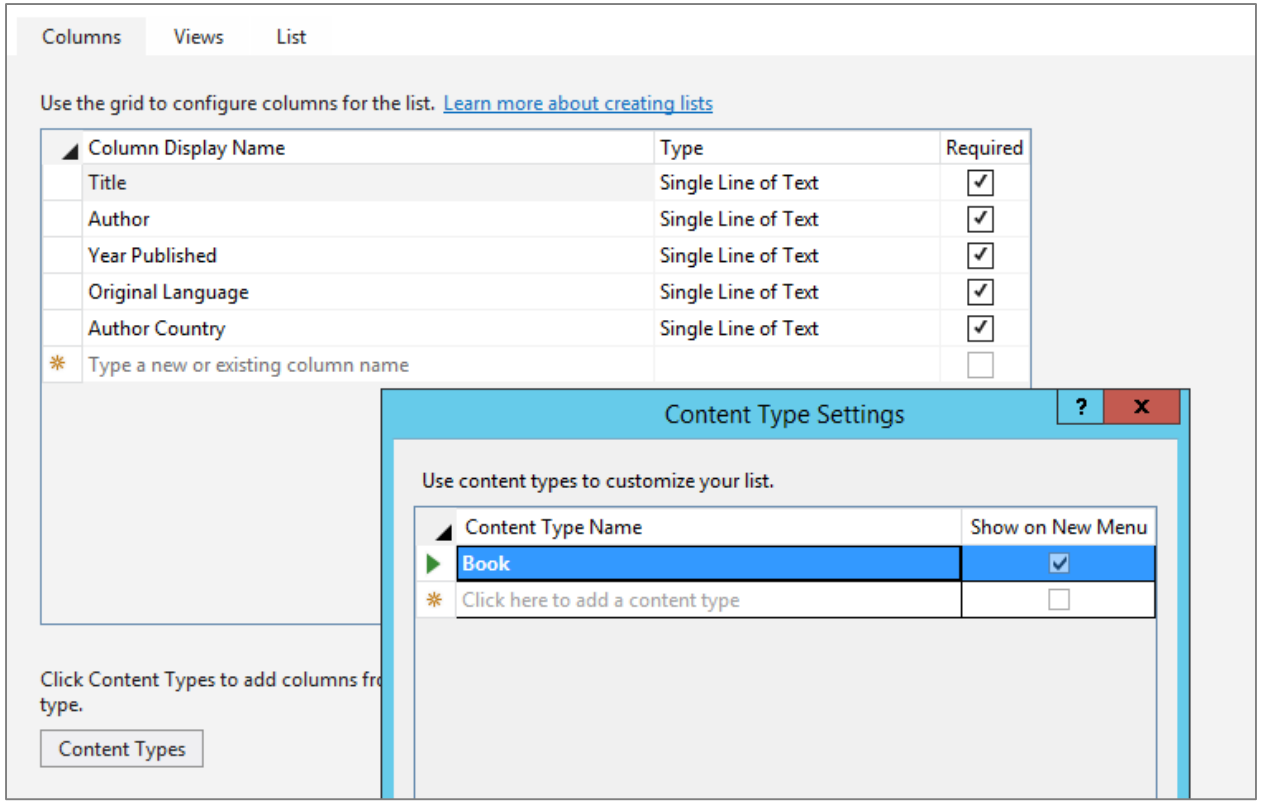

List Instance Element

List Template Element

List Template Definition

# Visual Studio List Designer

- List Designer abstracts away XML in Schema.xml
  - Used to add columns and content type support
  - Used configure list properties and add/modify views



# Provisioning Document Libraries

```
<?xml version="1.0" encoding="utf-8"?>
<ListInstance Id="Proposals"
              TemplateType="101"
              Title="Proposals"
              Description="Document Library for proposals"
              OnQuickLaunch="True"
              Url="Proposals">
</ListInstance>

<Module Name="TestData" List="101" Path="TestData" Url="Proposals" >
  <File Url="Adventure Works Proposal.docx" Type="GhostableInLibrary" />
  <File Url="Contoso Proposal.docx" Type="GhostableInLibrary" />
  <File Url="Wingtip Toys Proposal.docx" Type="GhostableInLibrary" />
</Module>


<Module Name="WordTemplate" List="101" Url="Proposals/Forms">
  <File Url="ProposalTemplate.dotx" Type="Ghostable" />
</Module>
```

DEMO

# Creating Lists

# Agenda

- ✓ Site Columns and Content Types
- ✓ Creating Lists and Document Libraries
- ➢ Updating Columns, Content Types and Lists
- Creating Event Receivers
- Creating Remote Event Receivers

# Feature Upgrade

- Used to version feature instances in production
  - Supported in SharePoint 2010 thru SharePoint 2016
- How does it work?
  - Feature definition is modified with Upgrade Actions
  - New feature definition pushed out using solution update
  - Feature instances queried and explicitly upgraded

# UpgradeActions

- Instructions for what to do during feature upgrade
  - **`ApplyElementManifest`** – used to process element manifest
  - **`CustomUpgradeAction`** – used to execute event handler
  - **MapFile** – used to remap existing file URL to new physical file
  - **AddContentTypeField** – used to add new column to existing content type

```
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
         Id="86689158-7048-4421-AD21-E0DEF0D67C81"
         Title="Wingtip Lead Tracker"
         Version="2.0.0.0" Scope="Web">
  <ElementManifests>
    <ElementManifest Location="elements.xml" />
    <ElementManifest Location="elements_v2.xml" />
  </ElementManifests>
  <UpgradeActions>
    <VersionRange BeginVersion="1.0.0.0" EndVersion="2.0.0.0">
      <ApplyElementManifests>
        <ElementManifest Location="elements_v2.xml"/>
      </ApplyElementManifests>
    </VersionRange>
  </UpgradeActions>
</Feature>
```

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <ListInstance Id="SalesLeads"
    TemplateType="105"
    Title="Sales Leads"
    Url="SalesLeads"
    OnQuickLaunch="TRUE" />
</Elements>
```

# Upgrading Content Types

- Use Upgrade Action named `<AddContentTypeField>`
  - Enables developers to easily upgrade content types

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/">
  <UpgradeActions>
    <VersionRange BeginVersion="1.0.0.0" EndVersion="2.0.0.0">
      <AddContentTypeField
              ContentTypeId="0x010600ADFF3A525118400BA1271AAAAB3A7F4B"
              FieldId="{41f2138d-f9d7-4bd8-a369-54054b016c22}"
              PushDown="TRUE"/>
    </VersionRange>
  </UpgradeActions>
</Feature>
```

# Triggering Feature Upgrade

- Updating solution do not trigger feature upgrade
  - Feature instances must be queried and upgraded
  - Typically done using a Windows PowerShell script

```
Add-PSSnapin Microsoft.SharePoint.Powershell -ErrorAction SilentlyContinue

$WebAppUrl = "http://intranet.wingtip.com"
$featureId = New-Object System.Guid -ArgumentList "86689158-7048-4421-AD21-E0DEF0D67C81"

$webApp = [Microsoft.SharePoint.Administration.SPWebApplication]::Lookup($WebAppUrl)
$features = $webApp.QueryFeatures($FeatureId, $true)

foreach($feature in $features){
  Write-Host "Updating feature in "$feature.Parent.Url
  $feature.Upgrade($true)
}
```

# Upgrading Content Types – Special Note

- Upgrading content types should be thoroughly tested in each scenario
- Strongly recommended to read the following resources:
  - Patterns & Practices: SharePoint Guidance http://msdn.microsoft.com/en-us/library/ff770300.aspx
  - P&P: SharePoint Guidance: Columns, Lists & Content Types http://msdn.microsoft.com/en-us/library/ff798404.aspx
  - MSDN Documentation: Updating Content Types http://msdn.microsoft.com/en-us/library/aa543504.aspx

# Agenda

- ✓ Site Columns and Content Types
- ✓ Creating Lists and Document Libraries
- ✓ Updating Columns, Content Types and Lists
- ➢ Creating Event Receivers
- • Creating Remote Event Receivers

# SharePoint Support for Events

- Server-Side Events
  - Supports Pre-Action & Post-Action Events
  - Event receivers created using Farm Solutions
  - Custom .NET code in DLL runs inside SharePoint
  - Not supported by new SharePoint App model
- Remote Events
  - Designed to give event-capabilities to SharePoint apps
  - Requires provider-hosted app
  - Not support with SharePoint-hosted app
  - Supports subset of events in server-side event model
  - Provides app lifecycle events as well

DEMO

# Server-Side Events

# Agenda

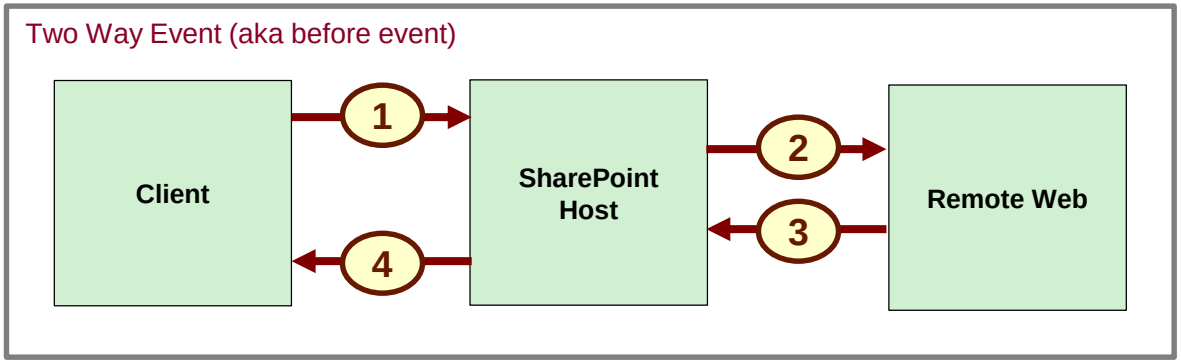
- ✓ Site Columns and Content Types
- ✓ Creating Lists and Document Libraries
- ✓ Updating Columns, Content Types and Lists
- ✓ Creating Event Receivers
- ➢ Creating Remote Event Receivers

# Event Handling Support in SharePoint 2016

- Classic server-side events for SharePoint solutions
  - Supported as early as SharePoint 2003
  - Based on ‘before’ events and ‘after’ events
  - Event handlers implemented using event receiver class in C#
  - Event handler assembly loads into SharePoint worker process
- New remote event infrastructure introduced for app model
  - Event handler code runs in remote web not in SharePoint host
  - SharePoint calls web service in remote web to trigger event
  - Set of supported remote events is subset of server-side events
  - Remote “before” events implemented as two-way events
  - Remote after events implemented as one-way events

# Remote “Before” Events

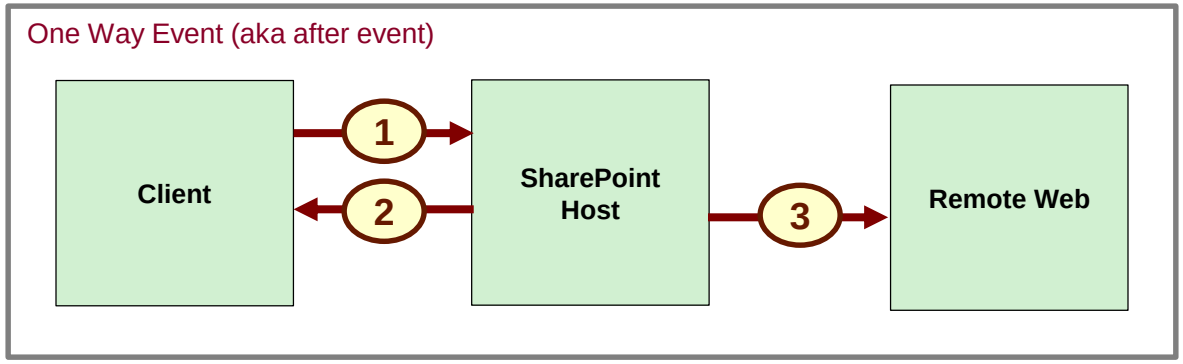
- Remote before events modeled as two-way events
  - Execution flow goes to remote web and then back to SharePoint
  - Client is blocked while event processing occurs in remote web
- Sample execution flow for two-way event
  1. Client attempts action which triggers an event (e.g. update item)
  2. SharePoint host calls to web service in remote web
  3. SharePoint host blocks until call returns from remote web
     *the key point is that two-way events block the SharePoint host and delay response back to user*
  4. SharePoint host commits action and returns to Client



# Remote “After” Events

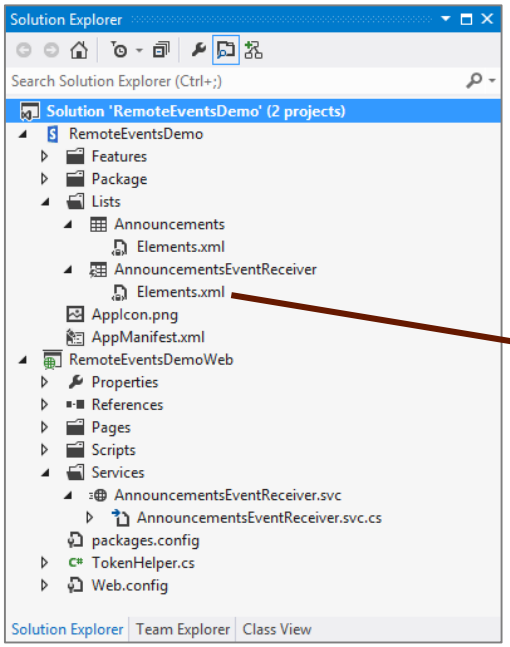
- Remote after events modeled as one-way events
  - Execution flow goes to remote web but does not return
  - Unlike before events, after events do not block client response
- Sample execution flow for one-way event
  1. Client attempts action which triggers an event (e.g. update item)
  2. SharePoint host commits action and returns to Client
  3. SharePoint host executes one-way WCF call on remote web

     *the key point is that one-way events do not block the SharePoint host nor delay response back to user*



# Registering Remote Events Receivers

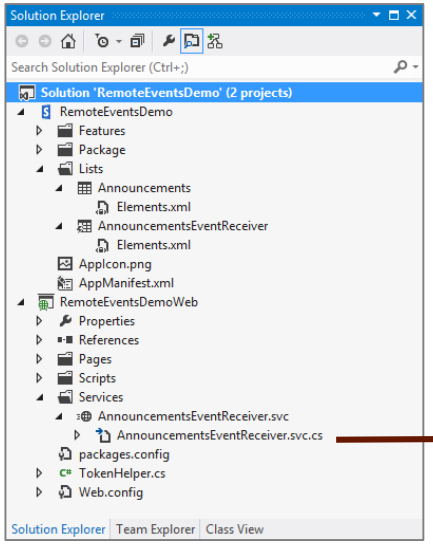
- Event receivers must be registered with SharePoint host
  - Registration can be declaratively for events occurring in app web
  - Registration for events occurring in host web requires code




```xml
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Receivers ListUrl="Lists/Announcements" >

    <Receiver>
      <Name>AnnouncementsEventReceiverItemAdding</Name>
      <Type>ItemAdding</Type>
      <SequenceNumber>10000</SequenceNumber>
      <Url>~remoteAppUrl/Services/AnnouncementsEventReceiver.svc</Url>
    </Receiver>

    <Receiver>
      <Name>AnnouncementsEventReceiverItemAdded</Name>
      <Type>ItemAdded</Type>
      <SequenceNumber>10000</SequenceNumber>
      <Url>~remoteAppUrl/Services/AnnouncementsEventReceiver.svc</Url>
    </Receiver>

  </Receivers>
</Elements>
```

# The Remote Event Receiver Entry Point

- Remote event receiver implemented with .svc file
  - Event receiver code written as C# code code-behind .svc file
  - Event receiver is class that implements **IRemoteEventService**
  - **ProcessEvent** method executes when two-way event is triggered
  - **ProcessOneWayEvent** method executes when one-way event is triggered




```
public class AnnouncementsEventReceiver : IRemoteEventService {

  // standard entry point for two-way events
  public SPRemoteEventResult ProcessEvent(SPRemoteEventProperties properties) {

    // TODO: add two-way event processing logic here

    // return SPRemoteEventResult back to SharePoint host
    SPRemoteEventResult result = new SPRemoteEventResult();
    return result;
  }

  // standard entry point for one-way events
  public void ProcessOneWayEvent(SPRemoteEventProperties properties) {

    // TODO: add one-way event processing logic here

  }
}
```

# SPRemoteEventProperties

- **SPRemoteEventProperties** parameter passed to event handlers
  - Provides you with contextual information about the current event
  - Allows you to read and update user input during before event
  - Makes it possible to perform validation in before events

```
// standard entry point for two-way events
public SPRemoteEventResult ProcessEvent(SPRemoteEventProperties properties) {

  properties.
```

EventType
GetHashCode
GetType
ItemEventProperties
ListEventProperties
SecurityEventProperties
ToString
UICultureLCID
WebEventProperties

SPRemoteItemEventProperties SPRemoteEventProperties.ItemEventProperties

# SPRemoteEventResult

- Two-way event handlers return **SPRemoteEventResult** object
  - **SPRemoteEventResult** object makes it possible to cancel action

```
// standard entry point for two-way events
public SPRemoteEventResult ProcessEvent(SPRemoteEventProperties properties) {
  SPRemoteEventResult result = new SPRemoteEventResult();
  result.
```

ChangedItemProperties
Equals
ErrorMessage
GetHashCode
GetType
RedirectUrl
Status — SPRemoteEventServiceStatus SPRemoteEventResult.Status
ToString

# Example Before Event with a List Item

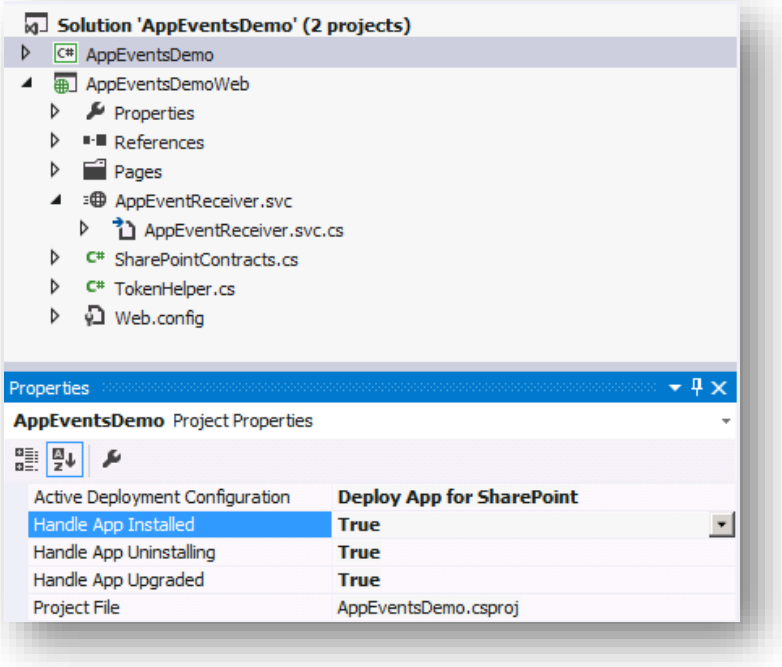
```
public SPRemoteEventResult ProcessEvent(SPRemoteEventProperties properties) {

  // create SPRemoteEventResult object to use as return value
  SPRemoteEventResult result = new SPRemoteEventResult();

  // inspect the event type of the current event
  if ( (properties.EventType == SPRemoteEventType.ItemAdding) ||
       (properties.EventType == SPRemoteEventType.ItemUpdating) ){

    // get user input to perform validation
    string title = properties.ItemEventProperties.AfterProperties["Title"].ToString();
    string body = properties.ItemEventProperties.AfterProperties["Body"].ToString();

    // perform simple validation on user input
    if (title.Contains("Google") || title.Contains("Apple") || title.Contains("NetScape")) {
      // cancel action due to validation error
      result.Status = SPRemoteEventServiceStatus.CancelWithError;
      result.ErrorMessage = "Title cannot contain inflammatory terms such as 'google', 'apple' or 'NetScape'";
    }

    // Process user input before it's added to the content database
    if (title != title.ToUpper()) {
      result.ChangedItemProperties.Add("Title", title.ToUpper());
    }
  }

  return result; // always return SPRemoteEventResult back to SharePoint host
}
```

# Example After Event with a List Item

```
public void ProcessOneWayEvent(SPRemoteEventProperties properties) {

  if ((properties.EventType == SPRemoteEventType.ItemAdded) ||
       (properties.EventType == SPRemoteEventType.ItemUpdated)) {

    // get column values of the new that has just been created
    string title = properties.ItemEventProperties.AfterProperties["Title"].ToString();
    string body = properties.ItemEventProperties.AfterProperties["Body"].ToString();

    // get information about the hosting list
    string list_title = properties.ItemEventProperties.ListTitle;
    Guid list_id = properties.ItemEventProperties.ListId;

    // get information about current user
    string user = properties.ItemEventProperties.UserDisplayName;

    // now do something with this info such as log it to an external database

  }
}
```

# App Lifecycle Events

- SharePoint app models support app events
  - App events for installation, upgrade and uninstall
  - Added to app project using property sheet
  - Implemented as a remote event receiver



DEMO

# Remote Event Receivers

# Summary

- ✓ Site Columns and Content Types
- ✓ Creating Lists and Document Libraries
- ✓ Updating Columns, Content Types and Lists
- ✓ Creating Event Receivers
- ✓ Creating Remote Event Receivers