DATA EXCHANGE FRAMEWORK

Migration LEGO for Sitecore developers

by Balázs Kerper

Agenda

- Introduction
- What is Data Exchange Framework?
- Building blocks
  - *Pipeline batches*
  - *Pipelines*
  - *Object locations*
  - *Pipeline steps*
  - *Data Access*
- Logging
- Pitfalls / Considerations

Introduction – Who am I / Who I’m not?

Who am I?

- Balázs Kerper
- .NET Developer – 5 years / Sitecore Developer – 2 years
- Various migration projects
- Email: balazs.kerper@gmail.com
- Twitter: @balazskerper

Who I’m not?

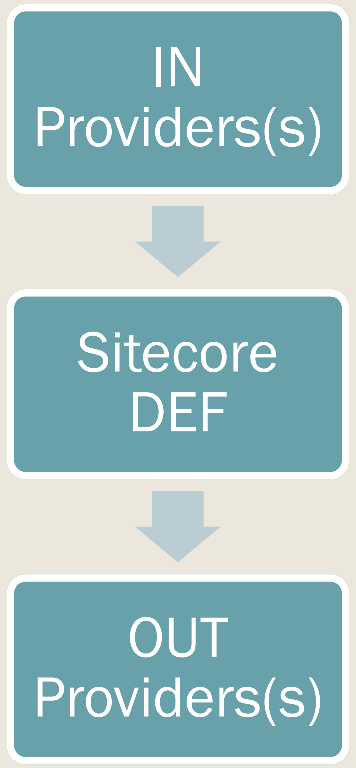
- Expert in Data Exchange Framework (DEF)

Introduction – then… Why?

- Migration / Synchronization of data is a common requirement
- …yet, DEF is not used widely (enough)
- DEF is:
  - *Powerful*
  - *Extensible*
  - *Fun!* ☺
- Sharing the experiences of the past ~1 year

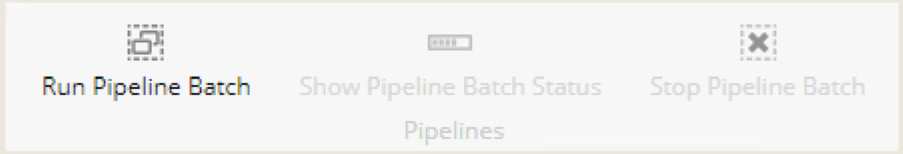
What is Data Exhange Framework (DEF)?

- Helps to „model processes to synchronise data between multiple systems”
- Manages these processes from within Sitecore
- Using default Sitecore capabilities:
  - *Items*
  - *Commands*
  - *Schedules*
- Providers can be customized, or new ones created
- Available at https://dev.sitecore.net/downloads
- Custom providers available from Sitecore Marketplace



Building Blocks – Pipeline Batches

- Contains one or more Pipelines, executes them in selection order
- Start / Stop / Status in Data Exchange tab
- Settings
  - *Runtime (e.g.: running out of process)*
  - *Logging (e.g.: log levels)*
  - *Administration (e.g.: enable/disable)*
  - *Summary (e.g.: start and finish times)*
- Can be run:
  - *Manually*
  - *Sitecore Task*
  - *Programmatically*



Building Blocks – Pipelines

- Models the order of the steps to synchronise data
- Contains a list of Pipeline steps, that are executed in order
- Can call other pipelines through specific Pipeline steps
- Example:

Pipeline 1 (runs once)	Pipeline 2 (runs for each line)
1. Read File Content to Iterable Data Setting 2. Iterate Lines and call Pipeline 2	1. Resolve Sitecore Item 2. Apply Mapping 3. Update Sitecore Item

Building Blocks – Object locations

- Pipelines contexts contain storage locations to store data temporarily
- Four object locations
  - *Pipeline Context Source (object / available from Parent)*
  - *Pipeline Context Target (object / available from Parent)*
  - *Pipeline Context Temp Storage (object / available from Parent)*
  - *Pipeline Context Iterable Data (IEnumerable<object>)*

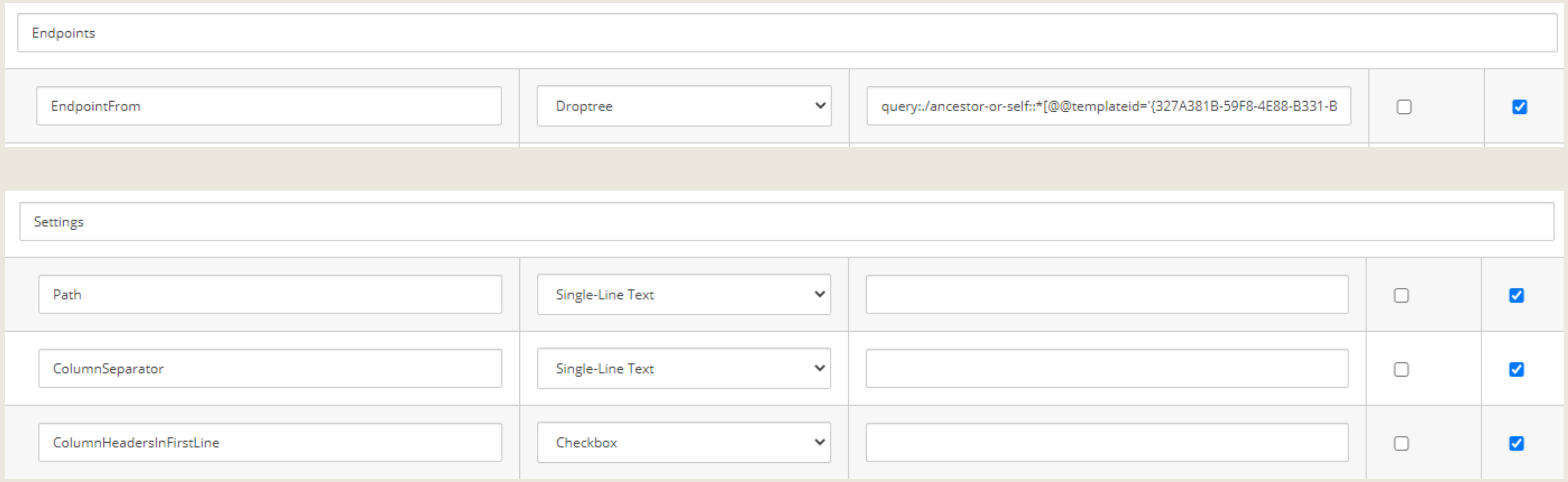
Pipeline 1 (runs once)	Pipeline 2 (runs for each line)
1. **Read File Content to Iterable Data Setting** - *each line is read to Pipeline Context Iterable Data* 2. **Iterate Lines and call Pipeline 2** – *iterates through lines in Pipeline Context Iterable Data and makes the line available Pipeline 2 Pipeline Context Source*	1. **Resolve Sitecore Item** – *resolves the ItemModel to Pipeline Context Target* 2. **Apply Mapping** – *Applies the mapping from the line in Pipeline Context Source to the ItemModel in Pipeline Context Target* 3. **Update Sitecore Item** – *Updates the Sitecore Item based on the ItemModel in Pipeline Context Target*

Building Blocks - Pipeline steps

- Represents a specific function in the pipeline
- 4 parts:
  - *Template/Model: represents the pipeline step and its settings in Sitecore*
  - *Converter: converts the Model to the Plugin*
  - *Plugin: contains the information of the Model in a DEF compatible format*
  - *Processor: the actual implementation of a logic*
- Pipeline steps can be Enabled/Disabled
- Example!

Building Blocks – Pipeline steps: Template

- Multipe Templates can be used as well
- Pipeline step & Endpoint templates:



Endpoints

EndpointFrom	Droptree	query:../ancestor-or-self::*[@@templateid='{327A381B-59F8-4E88-B331-B	☐	☑

Settings

Path	Single-Line Text		☐	☑
ColumnSeparator	Single-Line Text		☐	☑
ColumnHeadersInFirstLine	Checkbox		☐	☑

Building Blocks – Pipeline steps: Converter

- Converter for the Pipeline step & Endpoint settings
- Many available helpers
- Converters can support multiple Templates

```csharp
public class ReadTextFileStepConverter : BasePipelineStepConverter
{
    public ReadTextFileStepConverter(IItemModelRepository repository) : base(repository)
    {
        SupportedTemplateIds.Add(ReadTextFilePipelineStep.TemplateId.Guid);
    }

    protected override void AddPlugins(ItemModel source, PipelineStep pipelineStep)
    {
        pipelineStep.AddPlugin(new EndpointSettings
        {
            EndpointFrom = ConvertReferenceToModel<Endpoint>(source, ReadTextFileStepItemModel.EndpointFrom)
        });
    }
}
```

```csharp
public class TextFileEndpointConverter : BaseEndpointConverter
{
    public TextFileEndpointConverter(IItemModelRepository repository) : base(repository)
    {
        SupportedTemplateIds.Add(TextFileEndpoint.TemplateId.Guid);
    }

    protected override void AddPlugins(ItemModel source, Endpoint endpoint)
    {
        var settings = new TextFileSettings
        {
            ColumnHeadersInFirstLine = GetBoolValue(source, TextFileEndpointItemModel.ColumnHeadersInFirstLine),
            ColumnSeparator = GetStringValue(source, TextFileEndpointItemModel.ColumnSeparator),
            Path = GetStringValue(source, TextFileEndpointItemModel.Path)
        };
        endpoint?.AddPlugin(settings);
    }
}
```

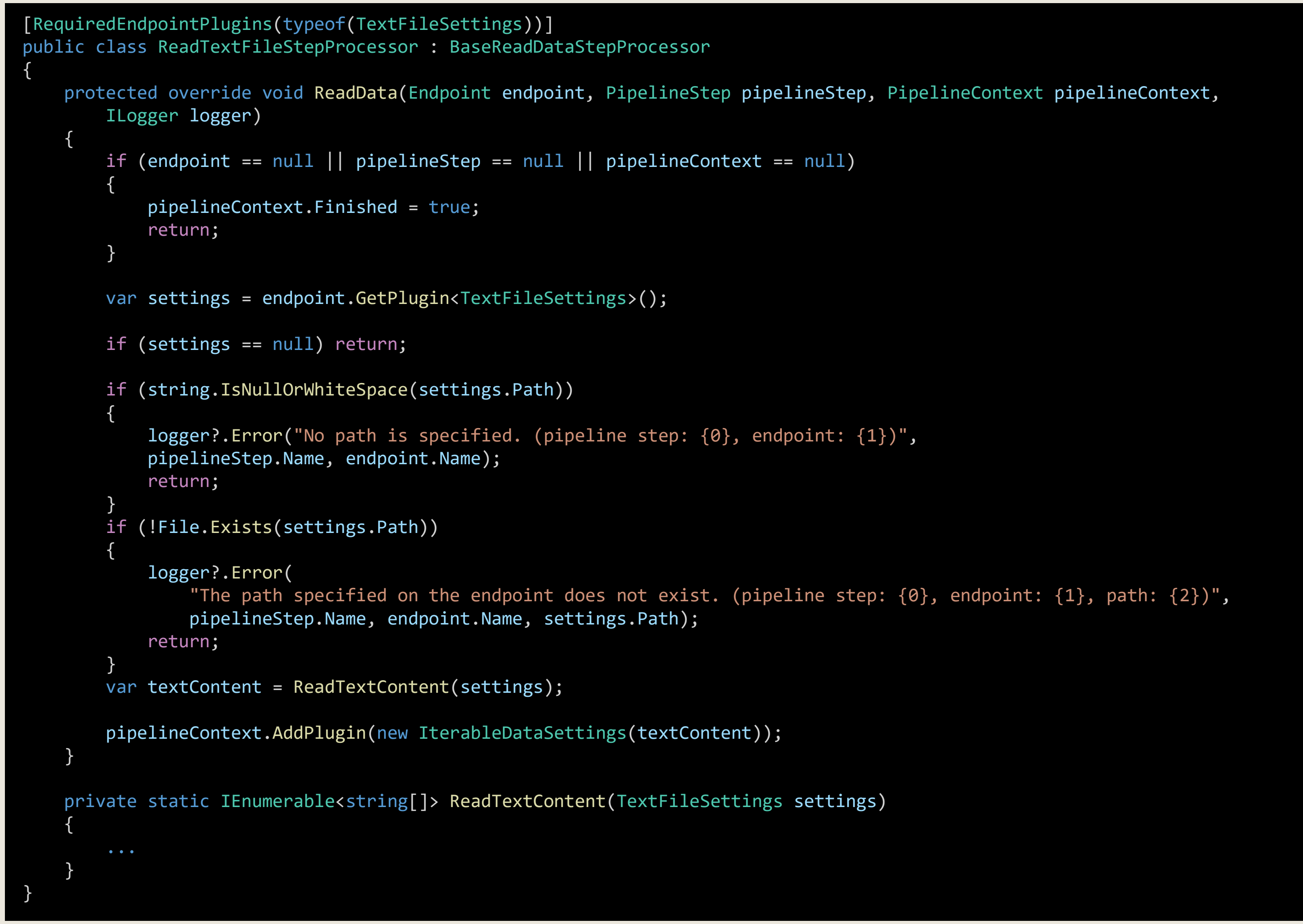
Building Blocks – Pipeline steps: Plugins

- Just simple classes inheriting from Iplugin
- Multiple Plugins can be used by a Processor

```
public class TextFileSettings : IPlugin
{
    public bool ColumnHeadersInFirstLine { get; set; }

    public string ColumnSeparator { get; set; }

    public string Path { get; set; }
}
```

Building Blocks – Pipeline steps: Processor

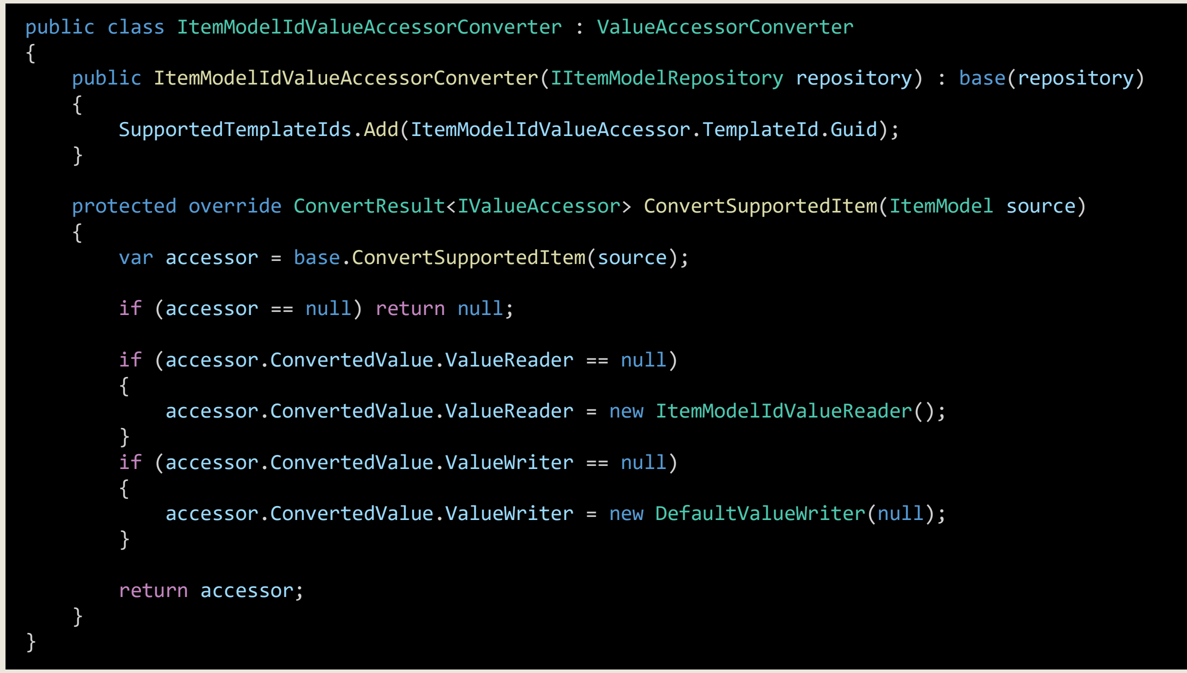
- Anything you can do programmatically, you can do here
- Specific BaseProcessors available for common operations
- Logging!



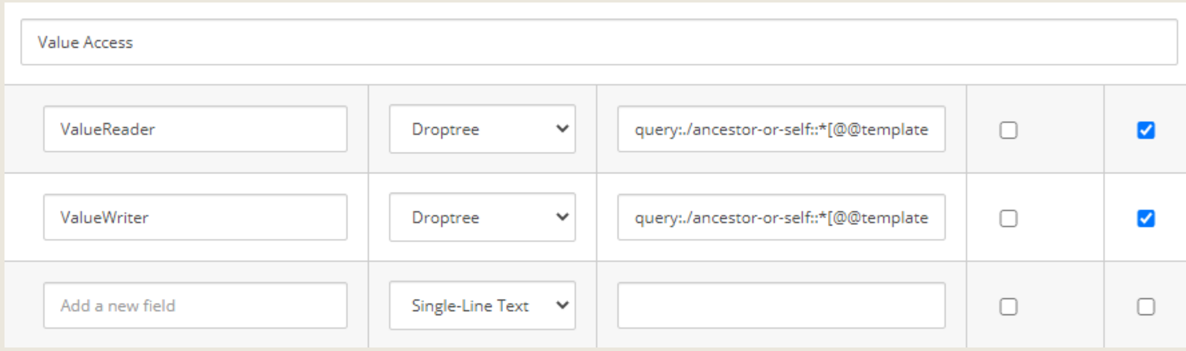
```
[RequiredEndpointPlugins(typeof(TextFileSettings))]
public class ReadTextFileStepProcessor : BaseReadDataStepProcessor
{
    protected override void ReadData(Endpoint endpoint, PipelineStep pipelineStep, PipelineContext pipelineContext,
        ILogger logger)
    {
        if (endpoint == null || pipelineStep == null || pipelineContext == null)
        {
            pipelineContext.Finished = true;
            return;
        }

        var settings = endpoint.GetPlugin<TextFileSettings>();

        if (settings == null) return;

        if (string.IsNullOrWhiteSpace(settings.Path))
        {
            logger?.Error("No path is specified. (pipeline step: {0}, endpoint: {1})",
            pipelineStep.Name, endpoint.Name);
            return;
        }
        if (!File.Exists(settings.Path))
        {
            logger?.Error(
                "The path specified on the endpoint does not exist. (pipeline step: {0}, endpoint: {1}, path: {2})",
                pipelineStep.Name, endpoint.Name, settings.Path);
            return;
        }
        var textContent = ReadTextContent(settings);

        pipelineContext.AddPlugin(new IterableDataSettings(textContent));
    }

    private static IEnumerable<string[]> ReadTextContent(TextFileSettings settings)
    {
        ...
    }
}
```

Data Access – Value Accessors (Sets)

- Value Accessors work like a .NET property: reads or writes the value
- Custom providers usually contain specific Value Accessors
- Value Accessor Converter: Sets default Value Reader and Value Writer
- Default Value Readers and Value Writers can be overridden
- Value Accessor Sets:
  - *collection of Value Accessors*
  - *helps organizing related Value Accessors*



```
public class ItemModelIdValueAccessorConverter : ValueAccessorConverter
{
    public ItemModelIdValueAccessorConverter(IItemModelRepository repository) : base(repository)
    {
        SupportedTemplateIds.Add(ItemModelIdValueAccessor.TemplateId.Guid);
    }

    protected override ConvertResult<IValueAccessor> ConvertSupportedItem(ItemModel source)
    {
        var accessor = base.ConvertSupportedItem(source);

        if (accessor == null) return null;

        if (accessor.ConvertedValue.ValueReader == null)
        {
            accessor.ConvertedValue.ValueReader = new ItemModelIdValueReader();
        }
        if (accessor.ConvertedValue.ValueWriter == null)
        {
            accessor.ConvertedValue.ValueWriter = new DefaultValueWriter(null);
        }

        return accessor;
    }
}
```

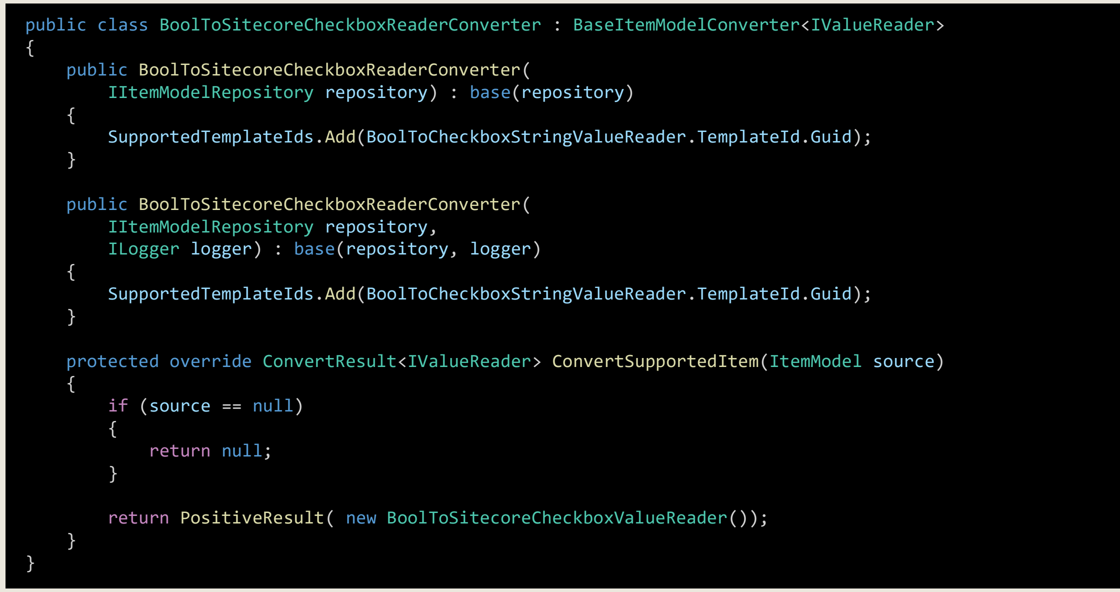


Value Access

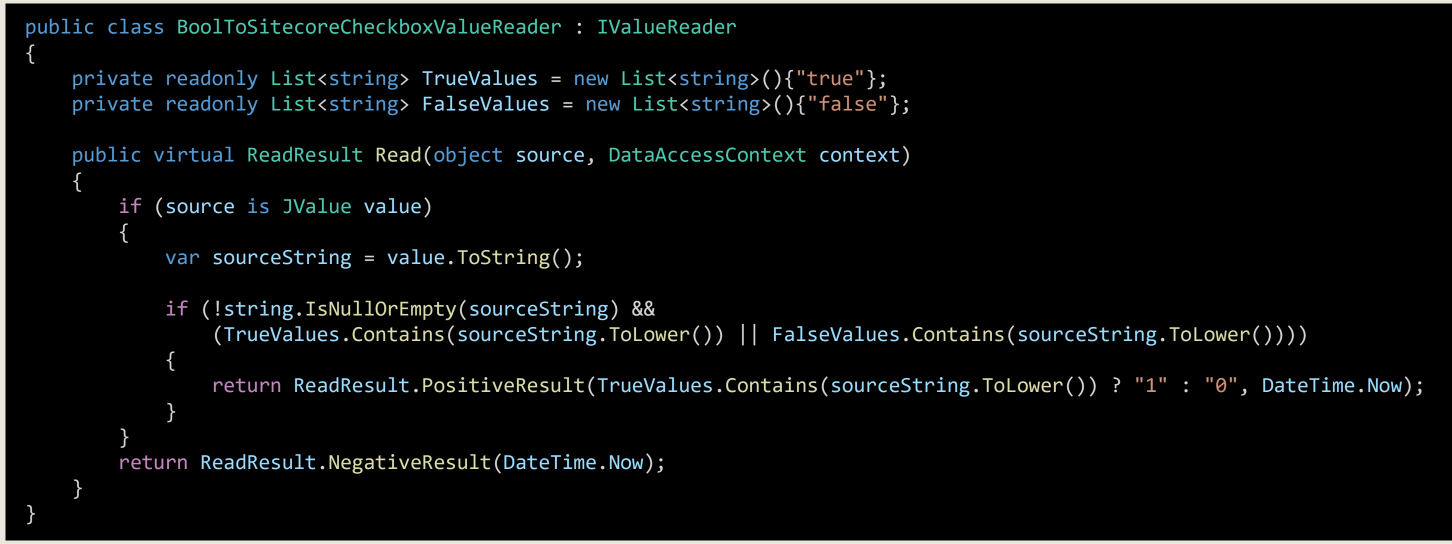
ValueReader	Droptree	query:./ancestor-or-self::*[@@template	☐	☑
ValueWriter	Droptree	query:./ancestor-or-self::*[@@template	☐	☑
Add a new field	Single-Line Text		☐	☐

Data Access – Value Readers/Writers

- Value Readers: property getters
- Value Writers: property setters
- Custom providers come with custom Value Readers and Value Writers



```csharp
public class BoolToSitecoreCheckboxReaderConverter : BaseItemModelConverter<IValueReader>
{
    public BoolToSitecoreCheckboxReaderConverter(
        IItemModelRepository repository) : base(repository)
    {
        SupportedTemplateIds.Add(BoolToCheckboxStringValueReader.TemplateId.Guid);
    }

    public BoolToSitecoreCheckboxReaderConverter(
        IItemModelRepository repository,
        ILogger logger) : base(repository, logger)
    {
        SupportedTemplateIds.Add(BoolToCheckboxStringValueReader.TemplateId.Guid);
    }

    protected override ConvertResult<IValueReader> ConvertSupportedItem(ItemModel source)
    {
        if (source == null)
        {
            return null;
        }

        return PositiveResult( new BoolToSitecoreCheckboxValueReader());
    }
}
```



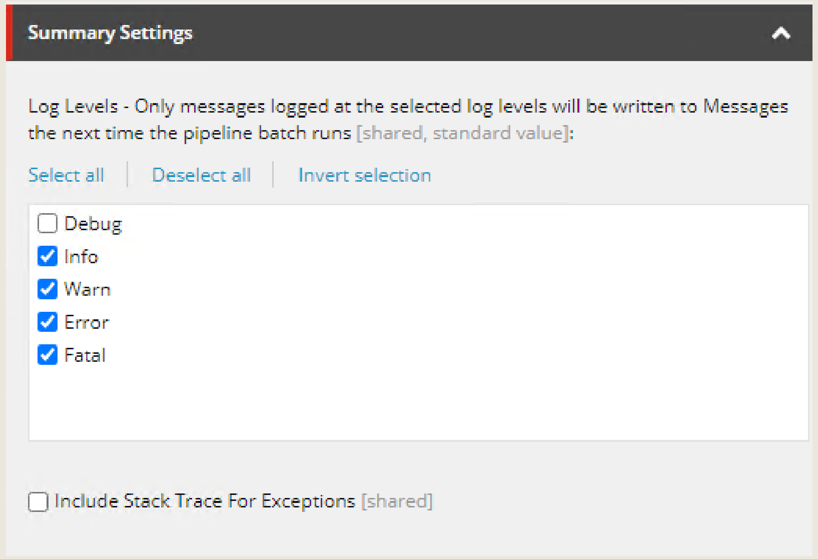
```csharp
public class BoolToSitecoreCheckboxValueReader : IValueReader
{
    private readonly List<string> TrueValues = new List<string>(){"true"};
    private readonly List<string> FalseValues = new List<string>(){"false"};

    public virtual ReadResult Read(object source, DataAccessContext context)
    {
        if (source is JValue value)
        {
            var sourceString = value.ToString();

            if (!string.IsNullOrEmpty(sourceString) &&
                (TrueValues.Contains(sourceString.ToLower()) || FalseValues.Contains(sourceString.ToLower())))
            {
                return ReadResult.PositiveResult(TrueValues.Contains(sourceString.ToLower()) ? "1" : "0", DateTime.Now);
            }
        }
        return ReadResult.NegativeResult(DateTime.Now);
    }
}
```

Data Access: Value Mapping (Sets)

- Value Mapping controls the mapping of a single value between objects
  - *Source Accessor: Value Accessor to read*
  - *Target Accessor: Value Accessor to write*
- Value Mapping Set
  - *Collection of mappings*
- Apply Mappings Pipeline Step
  - *Mapping Set: Value Mapping Set to apply*
  - *Source Location: location of the source object*
  - *Target Location: location of the target object*

Logging

- Logs can be found in /App_Data/logs/DataExchange
- Log levels to log can be set on Pipeline Batches. Example config:
  - *Testing/Dev: all levels => all information, huge amount of storage needed*
  - *Production: (Warn)/Error/Fatal => only error information*
- Use the correct log levels!
- CleanUpAgent in Production



Summary Settings

Log Levels - Only messages logged at the selected log levels will be written to Messages the next time the pipeline batch runs [shared, standard value]:

Select all | Deselect all | Invert selection

- [ ] Debug
- [x] Info
- [x] Warn
- [x] Error
- [x] Fatal

- [ ] Include Stack Trace For Exceptions [shared]

Pitfalls / Considerations

- Reusability vs. Complexity:
  - *Try to find a balance*
  - *Plan ahead*
- Resolving Items from indexes:
  - *Performance gains*
  - *Indexing needs to be controlled: pause index rebuild, and manually rebuild indexes when needed*
- Screenshot generation: disable if not needed, and changing items with a layout
- Removing Items
- https://doc.sitecore.com/developers/def/40/data-exchange-framework

?