The Standards

Table of content

Overview

- Standards provide a guiding light that we follow while writing code.
- Standards are born literally from blood, sweat and tears from lessons learned.
- Standards contain industry coding standards with a Signify flavor.
- They are the basis on which ALL Signify code reviews are done.

Why Code Review?

Having code reviews as part of our development workflow will bring different benefits to the entire team, such as:

1. Fewer bugs. Code reviews will decrease the amount of bugs that make it to production.
2. Better security. Code reviews make it easy to spot potential vulnerabilities and fix them before they make their way to production servers.
3. Better performance. Code reviews help spot performance issues and regressions.
4. Code quality. Things like readability, efficiency, and maintainability of your code may not always directly affect the end users, but they are tremendously important in the long run.
5. Knowledge sharing. Everyone on our team can learn from each other by reviewing each other’s code.

General

The following list of items indicate coding standards independent of the technology used.

- Comments should be added selectively if the code is not self-explanatory i.e. avoid over-commenting for code that is self-explanatory.
- Error handling should be done.
- Code should be maintainable code.
- No spelling mistakes.
- Removed unused code/comments during development.
- Code should be simplified / refactored.
- Translation resources should be used for non-dynamic text displayed to user.
- Variable names should explain the purpose of the variable.
- Enforce DRY (don't repeat yourself) principle
- KISS (Keep it simple stupid)
- The following general naming conventions must be followed unless otherwise specified
    - Microsoft Capitalization Conventions

C#

C#

Table of content

C#

General

Developers guide to debugging

(click on the image below)

- When dealing with objects, always check that they exist and content/elements are available

```
public IEnumerable<exSample> GetSamples(SignifyHRDAL dbContext)
{
  var samples = SampleHelper.GetSampleList();

  if (samples != null && samples.Count > 0)
  {
    //Do something
  }

  return samples;
}
```

- Format dates according to system standards (use SignifyTypeExtensions)
- Enums are singular
- Method names make sense

- Use the var keyword instead of long namespace.object.names if the type that is returned is clear from the variable initialisation.

```
public void UpdateSample()
{
  //Bad
  bool isActive = false;
  //Good
  var isActive = false;

  //Bad
  string sampleDescription = "This is the new description";
  //Good
  var sampleDescription = "This is the new description";
}
```

- Removed the old author from the method. Tracking of who made the change can be done in Git.
- Use object initializer instead of empty constructor, when setting properties.

```
public class Cat
{
    // Auto-implemented properties.
    public int Age { get; set; }
    public string Name { get; set; }

    public Cat()
    {
    }

    public Cat(string name)
    {
        this.Name = name;
    }
}
```

- Ensure that null checks are performed and handled, where necessary.
- Remember to remove unnecessary code.
- Lambda expressions – variables should make sense. Abbreviations can be used as long as it makes sense.
- Is foreach used in preference to the for(int i...) construct?

    - Foreach
    - For
- Use Translation Resources instead of hard coded text, especially for enums and meta data annotations. Also ensure that the translation resources are generated correctly using the admin page. Find steps here.
- Always prefix an interface with the letter I, for example ITransaction, IAuditable, etc.
- When naming an interface, where possible use adjective phrases, for example IRunnable, IAuditable, IPersistable, IDisposable, IComparable, IEnumerable, however, nouns can also be used such as ITransaction, IHttpModule, etc are allowed when deemed necessary.
- Use singular form in naming Enums, unless the enum represents bit-wise flags, where plural names should rather be used.
- When using generic type parameters in a generic type based classes or methods, use descriptive names such as IDictionary<TKey, TValue>, and prefix all generic type parameters with the letter T.
- Only use the letter T as a generic type parameter if it self-explanatory and usually the only generic type parameter, for example ICollection<T>, IEnumerable<T>, etc.
- When extending from the following classes, add the base class name as a suffix:
    - System.EventArgs, e.g. System.RepeaterEventArgs
    - System.Exception, e.g. System.SQLException
    - System.Delegate

C#

Statements, expressions and operators

References

- Statements
- Expressions and operators

Additional

- Ensure the proper use of extension methods and overloads. When you have to pass in a few null values for a method, consider making another overload. Also check that the null value is handled properly to avoid null exceptions. Should this value be null to start with OR Use a POCO object
- Ensure method arguments are validated and rejected with an exception if they are invalid

C#

Separation of concerns

The main concept here is **Single Responsibility** - "a class/method must have only one reason to change". This deals specifically with cohesion.

```
//Bad
public exSample UpdateSample(int id, string newDescription, int productId)
{
    //Method purpose is to return a sample
    var sample = SampleHelper.GetSample(id);

    //sample description is updated
    sample.description = newDescription;

  //A new product is also added the sample and the VAT calculated
  var product = ProductHelper.GetProduct(productId);
    product.VAT = product.TotalAmount * 14 / 100;
  sample.Products.Add(product);

    return samples;
}

//Good
//Method to update the sample (serves one purpose)
public exSample UpdateSample(int id, string newDescription, int productId)
{
    //Method purpose is to return a sample
    var sample = SampleHelper.GetSample(id);

    sample.description = newDescription;

    return samples;
}

//Method to add a product to the sample and do the product calculations where necessary (serves one purpose)
private exSample AddSampleProduct(exSample sample, int productId)
```

```
{
    //A new product is also added the sample and the VAT calculated
    var product = ProductHelper.GetProduct(productId);
    product.VAT = product.TotalAmount * 14 / 100;
    sample.Products.Add(product);

    return sample;
}
```

https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#single-responsibility

C#

Types

References

Build in reference types

Nullable reference types

SignifyTypeExtensions

TODO When are Types used? How are they used? Converting to other Types?

C#

Classes and Structs

C#

Interfaces

References

- Explicit Interface Implementation
- How to explicitly implement interface members
- How to explicitly implement members of two interfaces

C#

Comments

References

Commenting Conventions

C#

Arrays

References

- Arrays as Objects
- Single Dimensional Arrays
- Jagged Arrays
- Using `foreach` with Arrays
- Passing Arrays as Arguments

C#

Generics

References

- Generic Type Parameters
  - Constraints on Type Parameters
- Generic Classes
- Generic Interfaces
- Generic Methods
- Generics and Arrays

C#

Strings

References

- Working with Strings
- Formatting
  - Standard Numeric Format Strings
  - Custom Numeric Format Strings
  - Standard Date and Time Format Strings
  - Custom Date and Time Format Strings
  - Composite Formatting

Date Formatting (V8)

Ensure where ever a date is displayed to the user that it is done according to system setup / user preference:

```
public string CurrentDate
{
   get
   {
      return DateTime.Now.ToDateTime_Format();
   }
}
```

TODO : ToSafeString?

Any other extensions we can use/expand on

C#

Namespaces

Usage

Make use of `using` directives to enable improved readability and limit coding effort.

```
// Not making use of using directive
namespace MyTestProgram
{
   public class MyTestClass
   {
      private void DoSomething()
      {
         if(!System.IO.Directory.Exists("C:\\TestFolder\\");
            System.IO.Directory.CreateDirectory("C:\\TestFolder\\");

         var files = System.IO.Directory.GetFiles("C:\\MainFolder\\", "*.txt");
         var fileNames = new List<string>();

         foreach (var file in files)
         {
            fileNames.Add(System.IO.Path.GetFileName(file));
         }
      }
   }
}

// Making use of using directive
using System.IO;
namespace MyTestProgram
{
   public class MyTestClass
   {
      private void DoSomething()
      {
         if(!Directory.Exists("C:\\TestFolder\\");
```

```
            Directory.CreateDirectory("C:\\TestFolder\\");

        var files = Directory.GetFiles("C:\\MainFolder\\", "*.txt");
        var fileNames = new List<string>();

        foreach (var file in files)
        {
            fileNames.Add(Path.GetFileName(file));
        }
    }
  }
}
```

Aliasing

Use aliasing to prevent ambiguous references if different namespaces have objects with the same name.

```
using DataDomain = SignifyHR.Data.Domain;
using Domain = SignifyHR.Domain;
using SignifyHR.Helpers;

namespace SignifyHR.Learning
{
  public class HaveFun
  {
    public bool IsFun(int activityId)
    {
      var activity = Domain.Activity.TryFetch(activityId);
      var rules = DataDomain.Activity.SelectActivityPathwayRuleItemsList(new SessionHelper(), activityId);

      return activity != null && rules != null;
    }
  }
}
```

C#

Exception handling

References

Creating and Throwing Exceptions

Catch

Ensure the error is logged in a `catch`.
Actions returning a page, view or partial should return the appropriate error page.
Return the correct notification to user if elements are changed during execution or for AJAX/JSON responses.

```
// Log error
// Return correct error page/partial
public PartialViewResult PersonalDetails(int discussionId)
{
    try
    {
        return PartialView("_DiscussionPersonalDetails", new DiscussionPersonalDetailsViewModel(SessionHandler,
discussionId));
    }
    catch (Exception ex)
    {
        ErrorUtilities.LogException(ex);
        return PartialView("_Error");
    }
}



// Log error
// Return correct error notification
[HttpPost]
public ActionResult Approve(int id, DiscussionSection activeTab)
{
```

```
    try
    {
        ApproveDiscussion(id, DiscussionStatus.COMPLETED);
        return Json(new { success = true, message = "Successfully approved.", redirectUrl =
Url.EncryptedAction("Index", new { id, activeTab }) });
    }
    catch (Exception ex)
    {
        ErrorUtilities.LogException(ex);
        return Json(new { success = false, message = "There was a problem saving this section." });
    }
}



// Log error
// Return correct error notification
if (itemDisplaySetting == ItemDisplaySetting.Hide)
    HidePathwayStep(ref e);
else
{
    try
    {
        var step = pwStep.Fetch(sessionHandler, Int32.Parse(hfObjectID.Value));


    }
    catch (Exception ex)
    {
        ErrorUtilities.LogException(ex);
        lblReferenceDescription.Text = string.Format("{0}{1}", " Message: ", ex.Message);
    }
}
```

When to catch

Applying exceptions are usually done for the following cases:

- Logging of error and returning an error page/message:
  - Action called by user

    - Operation required for all following processes where normal `NULL` or content checks may not be sufficient.
- Logging of error, setting error message/property and continuing with next process

Shared libraries

SignifyTypeExtensions

- This library is used to to type casting for primitive types and has built-in error checking while performing type casting.
- This type casting is extended from String, DataTable and Object types.
- Examples:

```
var dateVariable = dt.ToDateTime(0, "ExpiredBeforeDate"); // extends DataTable type
var boolVariable = dt.ToBoolean(0, "ViewAssessmentTranscript");
var stringVariable = dt.ToString(0, "DocumentReferenceNumber");
var decimalVariable = this.ToDecimal();
var intVariable = this.ToInt32();
var doubleVariable = this.ToDouble();
var floatVariable = this.ToFloat();
```

TODO: Add example method for each type

SignifyControlLibrary

- This library extends certain controls inheriting from **System.Web.UI.WebControls** that are used regularly on web forms.
- Some properties are added to these controls that automate some of the functions that are normally performed on this control.
- Controls that are extended are

```
<asp:Repeater>
<asp:HiddenField>
<asp:TextBox>
<asp:Literal>
<asp:DropDownList>
```

- Examples:

```
<!--
AllowPaging - switch paging on or off for repeater
PageSize
        public enum PageSizes
        {
           Ten = 10,
           Twenty = 20,
           Fifty = 50,
           Hundred = 100
        }
NoResultsFoundMessage - override no results message
-->
<signify:SignifyRepeater ID="rptOverlapHistories" runat="server" AllowPaging="false"
NoResultsFoundMessage="No results found for your search criteria.">
<!--
Visible is by default false
-->
<signify:SignifyHiddenField ID="hfProcessObjectID" runat="server" Value="0" />
<!--
DisplayType
        public enum DisplayTypes
```

```
    {
        TextBox,
        TextArea,
        SearchBox
    }
    protected override void OnPreRender(EventArgs e)
    {
        base.OnPreRender(e);

        switch (_displayType)
        {
            case DisplayTypes.TextBox:
                this.TextMode = TextBoxMode.SingleLine;

                if (this.MaxLength == 0)
                    this.MaxLength = 500;
                break;
            case DisplayTypes.TextArea:
                this.TextMode = TextBoxMode.MultiLine;

                if (this.MaxLength == 0)
                    this.MaxLength = 2000;
                break;
            case DisplayTypes.SearchBox:
                this.TextMode = TextBoxMode.SingleLine;

                if (this.MaxLength == 0)
                    this.MaxLength = 200;
                break;
            default:
                break;
        }
    }
-->
<signify:SignifyTextBox MaxLength="50" ID="txtEmpNo" runat="server" DisplayType="SearchBox"
Width="170px"></cc:SignifyTextBox>
<!--
FormatDate - true/false (Should the text be Formatted to yyyy/mm/dd before rendering to the client?)
UseSession_DateFormat
EncodeHTML - true/false (Should the text be HtmlEncoded before rendering to the client?)
```

```
-->
<signify:SignifyLiteral ID="lPreferred" runat="server" Text="Preferred"></signify:SignifyLiteral>
<!--
Text - places i image before text in page heading
-->
<signify:SignifyPageDescription ID="pdOptMsg" runat="server" Text="When selected, steps in the pathway are copied but empty.<br>Each step must be configured e.g. for the Document step a document needs to be uploaded.<br>When not selected, a copy of the pathway structure is also made e.g. for a Document step, the document will already be available for download"></signify:SignifyPageDescription>
<!--
AddAllOption - Add an extra option named \"All\" to drop down list and returns a \"%\" in the value property, this properties can be changed
AllOptionText - Sets the text property for the \"All\" option
AllOptionValue - Sets the value property for the \"All\" option"
-->
<signify:SignifyDropDownList ID="ddlAppType" runat="server" AddAllOption="true" AllOptionText=""
AllOptionValue="" DataValueField="ObjectID" />
```

TODO: Single example per control, state what the defaults are

SignifyHR.Helpers

- Groups commonly used helpers used across all modules in Signify

Helpers available:

- ApiUtilities
- BooleanHelper
- CalendarControlHelper
- DateHelper
- DocumentUploadHelper
- EnumHelper
- ExceptionHelper
- FileHelper
- FunctionHelper
- GenerateTokenUtilities
- HorizontalCalendarControlHelper
- HttpExceptionHelper
- HttpResultHelper
- ImageUtilities
- IntegerHelper
- JsonCallHelper
- ModelHelper
- MvcUtilities
- NetworkHelper
- NetworkUitlities
- OrderHelper
- RoleUtilities
- ScormHelper
- SessionHelper
- SignifyHtmlUtilities
- SmsHelper
- StringHelper
- SvgHelper
- SystemStandards
- TransactionLogger
- XmlCallHelper
- VerticalCalendarControlHelper

TODO - description of helper and signature of 3 methods

Third-Party Integrations

SigniFlowUtilities

AdobeUtilities

QuestionMark

Rustici

ZoomConnect

Sharepoint

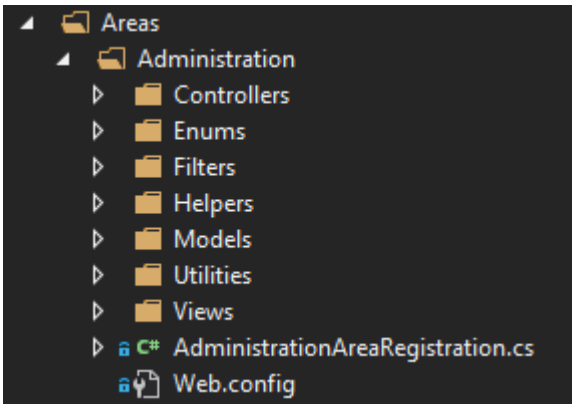
MVC

Table of contents

General

For a general guide on ASP.NET MVC, click on the following image

- Each functional domain being implemented must be added to its own Area e.g Administration
- The folder structure to use when creating a new MVC area is



- Each area must have the folders Controllers, Models and Views, the other folders are optional.
- The function of each folder is:
  - **Controller:** An interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.
  - **Views:** The View component is used for all the UI logic of the application.
  - **Models:** The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
  - **Filters:** The action filters that are attributes that can apply to a controller action or an entire controller that modifies the way in which the action is executed.

    - **Utilities:** The classes that receives a object and must do certain data modification or calculation custom to your domain are added here. These classes do not have access to the data domain and must be able to complete its action with the data provided e.g. Calculating the strength of a password
    - **Helpers:** Similar to utilities, helpers perform certain actions form or on data received. Helpers do have access to the data domain and can fetch additional data or perform data modification depending on the results of its calculation
    - **Enums:** Any enums only used in this domain are added here. It can be used for drop down population etc.
- Each area must have a web.config
    - It must define the namespace to include in the areas views
    - It must define the handlers and other configuration specific to that area

TODO : what happens in web.config? Add Example zip folder to General page

https://signature.signifyhr.co.za/books/tools/page/v8---mvc-development

MVC

Naming Conventions

The following naming convention must be followed when creating any of the MVC pattern sections. All names must be created using Pascal-Case.

Model

- The name is always a singularized representation of the database entity e.g. for the data base entity prsEmployees it must be prsEmployee
- When a model is required that is made up of SQL procedures for a specific domain of data, the name must represent the domain of data e.g. BadgesAndPoints is a model of different SQL objects returning data regarding badges and/ or points and nothing else.

Controller

- Every controller must be suffixed with the word Controller e.g HomeController
- Every controller must be placed within the ~/Controllers folder of its area

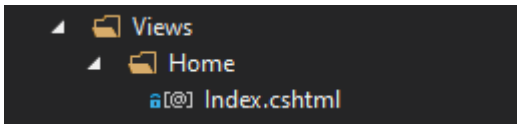
Attributes

All custom filter attributes must be named according to its function e.g. **BypassSession** Authorisation

View

- The view's name must represent the type of data it displays e.g. CreateEdit.cshtml
- The partialview's name must represent the type of data it displays and must always start with a underscore e.g. _OrderDetail.cshtml
- Every view must be placed within the ~/Views folder of its area matching the controllers name

  e.g. for the HomeController



- When the view is used by multiple controller it must be placed within the ~/Views/Shared folder of that area

View Models

- The name must always end with the postfix **ViewModel**
- The name can start with the action it is called from e.g. LoginViewModel
- The name can start with a descriptive name defining the type of data it encapsulates e.g. EmployeeDetailViewModel

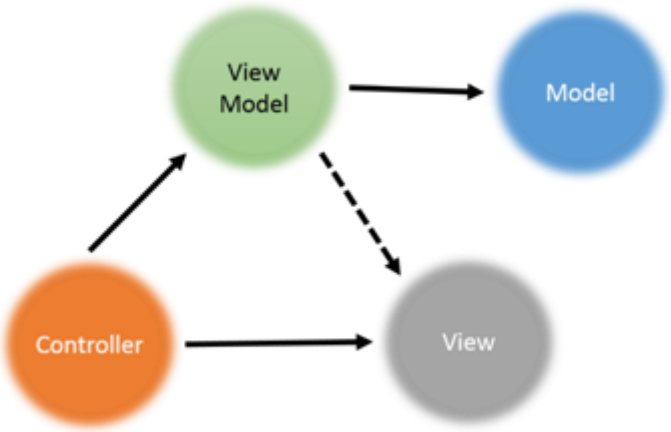
Routing

The registrations file for a areas route must be named {AreaName}AreaRegistration e.g LearningStoreRegistration.cs

Model/ Domain Model

Go To Naming Conventions

For information regarding models, click the images below



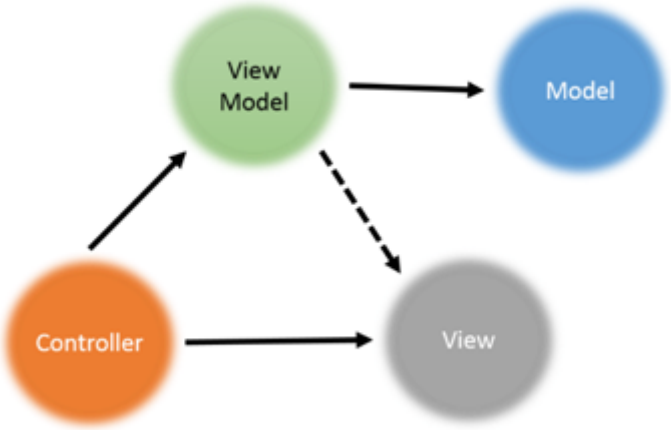
- Contain properties to represent specific data e.g EF Entities or SQL objects
- Contain business logic (e.g. validation rules) to ensure the represented data fulfills the design requirements
- May contain code for manipulating data. For example, a `SearchForm` model, besides representing the search input data, may contain a `search` method to implement the actual search. e.g. Domain Conventions
- The model may not contain any logic that is strictly required by the front end. This type of logic must rather be implemented in the View Model e.g. *build an html control based on business rules should be done the ViewModel instead of the Model.*
- Should not use `$_GET`, `$_POST`, or other similar variables that are directly tied to the end-user request.
- Should avoid embedding HTML or other presentational code.

MVC

ViewModel

Go To Naming Conventions

For information regarding view models, click the images below



- The class to group one or more models.
- It applies business logic on the results set from the models

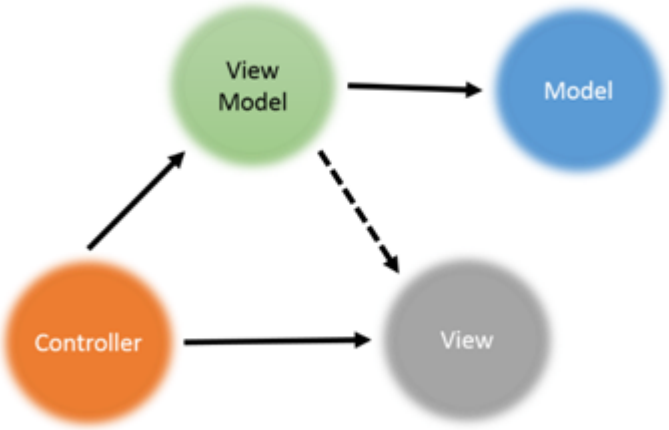
TODO : properties defined and properties inherited, private set properties, constructors

Example PathwayViewModel

View

Go To Naming Conventions

For information regarding razor views, click the images below



Responsible for presenting viewmodels, viewbags and other data structures in the format of the end user requirements

- Should mainly contain presentational code, such as HTML, to format and render data
- No code that performs explicit DB queries may be added to the view and must rather be placed in the view model.
- Should avoid direct access to `$_GET`, `$_POST`, or other similar variables that represent the end user request.
- May access properties and methods of controllers and view models directly only for presentational reasons

Reusing Views

Layouts

Common presentational areas (e.g. page header, footer) can be put in a layout view.

TODO - add link to Layout page in The Standards.

Partial Views

Views without a layout that reuse fragments of presentational code e.g. used in modals.

- Must be used if only parts of a page with data must be refreshed e.g. List with paging
- Represents code that can be used by multiple views in the Area.
- A partial must always start with an **underscore**.
- If the partial is only used inside the specific controller it must be in that specific folder. E.g. Details for LeaveGroupController will be in the LeaveGroup folder with the name: **_Details.cshtml**.
- A partial that shows items linked to the specific entity, must have the name of the collection linked to the entity, and must be in the folder of the entity. E.g. Leave Types linked to Leave Groups will be: **_LeaveTypes.cshtml** and it will be in the LeaveGroup folder.
- A partial that contains a list of items that can be linked to the entity must follow the convention in 3. and must follow these naming conventions:
  1. If it is a multiple link list (with the delink, link and in use buttons): _LeaveTypes **LinkMultiple**.cshtml
  2. If it is a picker list that populates a textbox on the parent page (you can only link one item at a time): _LeaveTypes**SelectSingle**.cshtml
- A partial with a list of items that is generic and shared over multiple controllers must be in the Shared folder.
  1. If it is shared only in the current Area, it must be in the Area's shared folder: ***App/Areas/Module/Views/Shared***
  2. If it is shared over multiple areas it should be in the global shared folder: ***App/Views/Shared***
- A partial that is used for navigation should follow these naming conventions: _LeaveTypes **SideNav**.cshtml is a partial for Leave Groups that is used to navigate between Leave Types linked to a Leave Group
- Exceptions:
  1. When using a partial in a one view only but for AJAX posting i.e. an action returns **PartialViewResult** in controller
- If the partial is shared between views, the javascript needs to reside on the partial itself.
- Javascript shared among multiple partials with the same parent view should reside on the parent view.

Helpers Classes

In views we often need some code snippets to do tiny tasks such as formatting data or generating HTML tags.

- Small segments can be placed directly in the view, but larger segments must be moved to a class file
- An example of a helper and its use in a view

The method in the helper class

```
@helper StepLink(string name, string icon, string actionText, string actionIcon, string onclick = null, string href = null, string target = null) {
    // Must have action text as well as an onclick or an href
    var clickable = !String.IsNullOrWhiteSpace(actionText) && (!String.IsNullOrWhiteSpace(onclick) || !String.IsNullOrWhiteSpace(href));

    <div class="step-link">
        <div class="step-link__name">
            @System.Web.Mvc.MvcHtmlString.Create(name)
        </div>

        <a class="step-link__icon" onclick="@onclick" href="@href" target="@target" @if (!clickable) {<text>style="cursor: default"</text>}>
            <i class="fa fa-@icon"></i>
        </a>

        @if (clickable)
        {
            <a class="step-link__action primary-bg" onclick="@onclick" href="@href" target="@target">
                <i class="fa fa-@actionIcon"></i> <span>@actionText</span>
            </a>
        }
    </div>
}
```

The implementation of the class and method in the view

```
<div class="text-center">
    @PathwayHelpers.StepLink(Model.QuestionMarkAssessment.AssessmentId, "question-circle", "Launch", "paper-plane", Model.Url)
</div>
```

TODO: add examples from v8 mvc page

Ensure list data bound during post (use for instead of for each)

Difference between Html.Form and Ajax.Form

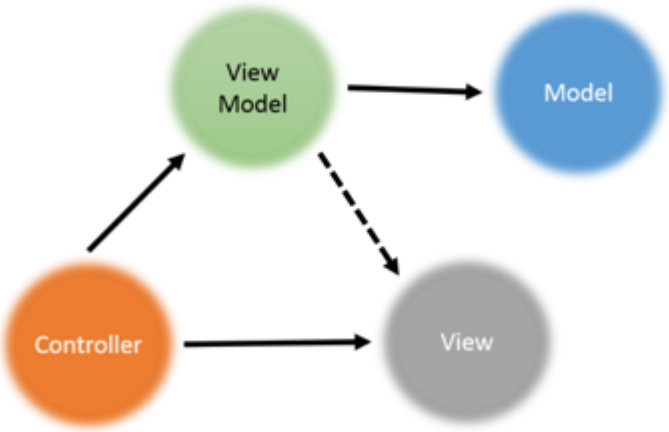
Use of hidden field variables and where they should be placed in a view

MVC

Controller

Go To Naming Conventions

For information regarding Controllers, click the images below



- Each controller must implement the IController interface directly or indirectly, this must be done using the BaseController

```
namespace SignifyHR.Areas.Portal.Controllers
{
   public class HomeController : BaseController
   {
      [UserTranslationFilter]
      public ActionResult Index(PortalTabs? activeTab)
      {
         var model = new HomeViewModel(SessionHandler, activeTab)
         {
            Title = TranslationResources.PageHeadingsLmsPortal,
         };

         return View(model);
      }
 }
```

- TODO from v8 - MVC Examples

https://signature.signifyhr.co.za/books/tools/page/v8---mvc-development/edit

When to use ActionResult and JSONResult

Link to exception

AllowGet

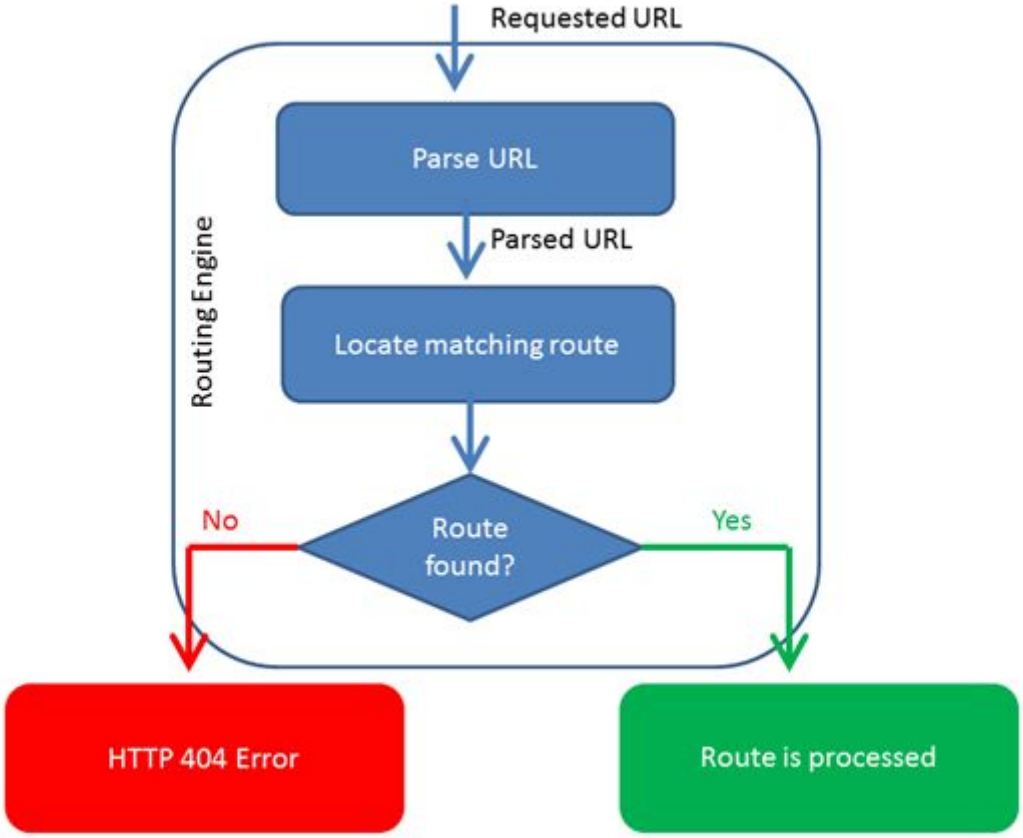
Follow C# standards

MVC

Routing

Go To Naming Conventions

For a guide regarding routing, click on the image below



Area specific routing is implemented in the Area Registration cs file and can have custom routing as required by the area e.g. LearningStoreAreaRegistration.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace SignifyHR.Areas.LearningStore
{
```

```
    public class LearningStorereaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "LearningStore";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
             name: "LearningStore.Pages",
             url: "LearningStore/{id}/{slug}",
             defaults: new { controller = "Page", action = "Index", slug = UrlParameter.Optional },
             constraints: new { id = @"\d+" },
             namespaces: new[] { "SignifyHR.Areas.LearningStore.Controllers" }
            );

            context.MapRoute(
             name: "LearningStore.DEFAULT",
             url: "LearningStore/{controller}/{action}",
             defaults: new { controller = "Home", action = "Index" },
             namespaces: new[] { "SignifyHR.Areas.LearningStore.Controllers" }
            );

        }
    }
}
```

MVC

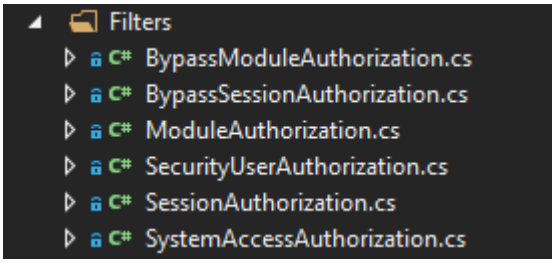
Attributes

Go To Naming Conventions

The ASP.NET MVC framework supports four different types of filters executed in the following order:

1. **Authorisation filters** – Implements the IAuthorizationFilter attribute.
2. **Action filters** – Implements the IActionFilter attribute.
3. **Result filters** – Implements the IResultFilter attribute.
4. **Exception filters** – Implements the IExceptionFilter attribute.

The common authorisation filters that is used in Signify are



- **BypassModuleAuthorisation:** The MVC authorization filter that enables one to by-pass module authentication.
- **BypassSessionAuthorisation:** The MVC authorization filter that enables one to by-pass session authentication.
- **ModuleAuthorisation:** Extended authorisation that determines if the user consists of a session.
- **SecurityUserAuthorisation:** Authorize the user against roles.
- **Session Authorisation:** Extended authorisation that determines if the user consists of a session, implemented in the base controller
- **SystemAccessAuthorisation:** Authorize the user on access to the module and the active status of the module.

Example of use

Applying a filter on a single method in the controller

```
[BypassSessionAuthorization]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult Confirmation(Guid token)
{
    try
    {
        var decryptedToken = sysAccessRequest.TokenDataWrapper.Decrypt(token);

        if (decryptedToken != null)
        {
            return View(new AccessRequestConfirmationViewModel(decryptedToken.sysUserRegistrationId, decrypte
        }
        else
        {
            throw new Exception("The request token could not be deserialized.");
        }
    }
    catch (Exception ex)
    {
        ErrorUtilities.LogException(ex);
        throw new Exception("The request token could not be decrypted or deserialized.", ex.InnerException);
    }
}
```

Applying a filter on the controller as a whole

```
namespace SignifyHR.Areas.Security.Controllers
{
    [BypassSessionAuthorization]
    [BypassModuleAuthorization]
    0 references | Martinique Lourens, 62 days ago | 7 authors, 11 changes | 2 work items
    public class AccountController : SignifyHR.Controllers.BaseController
    {
```

Extensions and Tools

Logging

Using NLog

```
public class HomeViewModel : PortalMasterModel
{
    private static readonly Logger _logger = LogManager.GetCurrentClassLogger();
    private static readonly Logger _lmsLogger = LogManager.GetLogger("LMS");

    public HomeViewModel()
    {
        // Using the class name as the name of the logger
        _logger.Info("Initializing HomeViewModel");

        // Using a new logger instance with a property automatically added
        _lmsLogger.WithProperty("ActiveTab", activeTab).Info("Initializing HomeViewModel");

        // Setting a property permanently on a logger
        _logger.SetProperty("Key", "Value");
        _logger.Info("Testing EventProperties");

        // Adding properties automatically in the message
        _logger.Info("Testing dynamic EventProperty Key={PropertyName}", "PropertyValue");

        // Logging an exception
        _logger.WarnException("Exception", new System.Exception("Example Exception"));
    }
}
```

Example of log levels

Level	Message	Logger	Properties	Callsite	Exception

Info	Initializing HomeViewModel	SignifyHR.Areas.Portal.ViewModels.HomeViewModel		SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Info	Initializing HomeViewModel	LMS	ActiveTab=TrainingRequirements	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Info	Testing EventProperties	SignifyHR.Areas.Portal.ViewModels.HomeViewModel	Key=Value	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Info	Testing dynamic EventProperty Key="PropertyValue"	SignifyHR.Areas.Portal.ViewModels.HomeViewModel	PropertyName=PropertyValue|Key=Value	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Warn	Exception	SignifyHR.Areas.Portal.ViewModels.HomeViewModel	Key=Value	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	System.Exception: Example Exception

Log Levels

Level	Typical Use
Fatal	Something bad happened; application is going down
Error	Something failed; application may or may not continue
Warn	Something unexpected; application will continue
Info	Normal behavior like mail sent, user updated profile etc.
Debug	For debugging; executed query, user authenticated, session expired
Trace	For trace debugging; begin method X, end method X

Rules

A rule is a logger element with the following attributes:

- name – logger name filter - may include wildcard characters (* and ?)
- minlevel – minimal level to log

- maxlevel – maximum level to log
- level – single level to log
- levels - comma separated list of levels to log
- writeTo – comma separated list of targets to write to
- final – no rules are processed after a final rule matches
- enabled - set to false to disable the rule without deleting it
- ruleName - rule identifier to allow rule lookup with Configuration.FindRuleByName and Configuration.RemoveRuleByName.

NLog.config

```
!--
   Create a rule to only log entries from the LMS HomeViewModel logger,
   using the logDatabase target, and applicable on log levels Info and above
-->
<logger name="SignifyHR.Areas.Portal.ViewModels.HomeViewModel" minlevel="Info" writeTo="logDatabase"
enabled="true" final="true" />
```

MVC

API Conventions

Propagate errors. If an error occurred, return a 500. The status code should tell you what happened, not the result of the call itself. Follow the DefaultApiConventions, which state that the action should return void and produce a 200, 404, or 400 status. In MVC these rules don't have to be followed as strictly, but with the APIs it is non-negotiable for standardisation.

```
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesDefaultResponseType]
[ApiConventionNameMatch(ApiConventionNameMatchBehavior.Prefix)]
public static void Delete(
[ApiConventionNameMatch(ApiConventionNameMatchBehavior.Suffix)]
[ApiConventionTypeMatch(ApiConventionTypeMatchBehavior.Any)]
object id) { }
```

https://github.com/dotnet/aspnetcore/blob/master/src/Mvc/Mvc.Core/src/DefaultApiConventions.cs

Entity Framework

Table of contents

General

- All methods accept and use ISessionHandler

```
using SignifyHR.Core;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;

namespace SignifyHR.Data.Domain
{
   public partial class exSample : IAuditable
   {
      protected static IQueryable<exSample> ValidSamples(SignifyHRDAL dbContext, EagerLoadParameters
eagerLoadParms = null)
      {
         var samples = dbContext.exSamples.AsQueryable();

         if (eagerLoadParms != null)
         {
            if (eagerLoadParms.IncludeSampleDocuments)
               samples = samples.Include(item => item.exSampleDocuments);

            if (eagerLoadParms.IncludeSampleComments)
               samples = samples.Include(item => item .exSampleComments);
         }

         return samples;
      }

      public static IEnumerable<exSample> FetchAll(ISessionHandler sessionHandler, SearchParameters
searchParms, EagerLoadParameters eagerLoadParms = null)
      {
         using (var dbContext = new SignifyHRDAL(sessionHandler))
```

```
        {
            var results = ValidSamples(dbContext, searchParms, eagerLoadParms);

            return results.OrderByDescending(item => item.Id)
                .Skip(searchParms.Skip)
                .Take(searchParms.Take)
                .ToList();
        }
    }
  }
}
```

- Eager loading used responsibly

```
using SignifyHR.Core;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;

namespace SignifyHR.Data.Domain
{
   public partial class exSample : IAuditable
   {
      #region Eager Load Parameters

      public class EagerLoadParameters : BaseSearchParameters
      {
         public bool IncludeSampleDocuments { get; set; }
         public bool IncludeSampleComments { get; set; }
      }

      #endregion

      #region Protected Methods

      protected static IQueryable<exSample> ValidSamples(SignifyHRDAL dbContext, EagerLoadParameters
eagerLoadParms = null)
      {
```

```
        var samples = dbContext.exSamples.AsQueryable();

        if (eagerLoadParms != null)
        {
            if (eagerLoadParms.IncludeSampleDocuments)
                samples = samples.Include(item => item.exSampleDocuments);

            if (eagerLoadParms.IncludeSampleComments)
                samples = samples.Include(item => item .exSampleComments);
        }

        return samples;
      }
   }
}
```

- IQueryable declared as **protected**

```
protected static IQueryable<exSample> ValidSamples(SignifyHRDAL dbContext, EagerLoadParameters eagerLoadParms = null)
protected static IQueryable<exSample> FilterSamples(SignifyHRDAL dbContext, SearchParameters searchParms, EagerLoadParameters eagerLoadParms = null)

public static exSample Fetch(ISessionHandler sessionHandler, int id, EagerLoadParameters eagerLoadParms = null)
public static exSample TryFetch(ISessionHandler sessionHandler, int id, EagerLoadParameters eagerLoadParms = null)
```

- Create a POCO Object when you want to call a **Stored Procedure** or **View** from Entity Framework

```
using SignifyHR.Core;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;

namespace SignifyHR.Data.Domain
```

```
{
   public partial class exSample : IAuditable
   {
□□public class POCOPreview
      {
         public int ExampleId { get; set; }
         public string ExampleDescription { get; set; }
         public bool ExampleBool { get; set; }
      }
   }
}
```

- Search parameter array used

```
using SignifyHR.Core;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;

namespace SignifyHR.Data.Domain
{
   public partial class exSample : IAuditable
   {
      #region Search Parameters

      public class SearchParameters : BaseSearchParameters
      {
         public int? SomeId { get; set; }
         public string Description { get; set; }
         public bool IsUsed { get; set; }
      }

      #endregion

      #region Protected Methods
```

```
        protected static IQueryable<exSample> FilterSamples(SignifyHRDAL dbContext, SearchParameters searchParms, EagerLoadParameters eagerLoadParms = null)
        {
            var result = ValidSamples(dbContext, eagerLoadParms);

            if (searchParms!= null)
            {
                if (!String.IsNullOrWhiteSpace(searchParms.Description))
                    result = result.Where(item => item.Description.ToLower().Contains(searchParms.Description.ToLower()));

                if (searchParms.SomeId.HasValue)
                    result = result.Where(item => item.SomeId == searchParms.SomeId.Value);

                if (searchParms.IsUsed.HasValue)
                    result = result.Where(item => item.IsUsed == searchParms.IsUsed.Value);
            }

            return result;
        }

        #endregion

        #region Public Methods

        public static IEnumerable<exSample> FetchAll(ISessionHandler sessionHandler, SearchParameters searchParms, EagerLoadParameters eagerLoadParms = null)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                var results = FilterSamples(dbContext, searchParms, eagerLoadParms);

                return results.OrderByDescending(item => item.Id)
                        .Skip(searchParms.Skip)
                        .Take(searchParms.Take)
                        .ToList();
            }
        }

        #endregion
```

```
    }
}
```

- No DateTime values are passed to the database (different servers = different time = different results).
- Domain Convention were followed.
- First, FirstOrDefault, Single, SingleOrDefault used for the correct purpose:
    - **First** - when one or more entities may be returned but only the first one is used (Remember to use OrderBy to return the correct entity)
    - **FirstOrDefault** - when none, one or more entities are returned, but only the first one is used (Remember to use OrderBy to return the correct entity).
    - **Single** - when only one entity will ALWAYS be returned
    - **SingleOrDefault** - when one or no entities are expected

TO DO : Add database date fetch method

Use of sp's vs LINQ and limitations

Do not use Views and SP's directly in entity framework, use POCO classes to map objects

Naming conventions

Methods for EF are written in Pascal Case

Examples of naming conventions that need to be followed for EF Methods:

Method
Fetch
TryFetch
FetchAll
FetchAllBy<Association>
TryFetchFirst
Create
Update
Delete

TODO: Examples, when is a postfix added to the above method names or not?

Which method is used for paging?

Domain convention

Methods

Method	Parameters	Linq / Entity Operation	Return Type	State	Dal Sync Mode
Fetch	(id, optional filter params, eagerLoaded = false / eagerLoadParms = null)	Single	T	Static	Default
TryFetch	(id, optional filter params, eagerLoaded = false / eagerLoadParms = null)	SingleOrDefault	T?	Static	Default
FetchAll	(optional filter params)	Where	IEnumerable<T> , IPagedList<T>	Static	Default
FetchAllBy<Association>	(<Association>Id , optional filter params, eagerLoaded = false / eagerLoadParms = null)	Where	IEnumerable<T> , IPagedList<T>	Static	Default
TryFetchFirst	None	FirstOrDefault	T?	Statuc	Default
Create	None	AddObject(this)	Void	Non-Static	Sync
Update	None	Attach(this) / Modified Entity State	Void	Non-Static	Sync
Delete	None	Attach(this) / Deleted Entity State	Void	Non-Static	Sync

Queryable<T>{All}{By<Association>	SignifyHRDAL	IQueryable	IQueryable<T>	Static	Default
Is<Condition>, Can<Condition>, Has<Condition>, etc.	None / Property not method	Boolean Check	Bool	Non-Static	Default
<Enum>Type	None / Property not method	<T>TryParseEnum	<T>	Non-Static	Default
CreateEdit	Call Create / Update separately from method. Is Wrapper Method	Call Methods Separately	Void	Non-Static	Sync

Meta Data

Meta data is used when rendering MVC Razor views and are defined in the domain if display name differs from specified name in base table, values are required and/or format of value.

Table 2 : Meta Data Properties

Property	Reason	Example
Required	Mark value as required and performs validation when submitting from form on view. An error message can be defined to display if validation fails.	[Required(ErrorMessage = "The {0} field is required")]
Display(Name=<string>)	Text to display when using DisplayFor() on views.	[Display(Name = "Start Date")]
DataType(<DataType>)	Formats and validate input according to specified type.	[DataType(DataType.DateTime)]

Examples

Template

Below is an example of a basic domain. This can be used as a template when creating new domains; copy and paste the code, replace all the required words (exSample, sample, smpl) according to applicable entity name.

Note: For domains with a single eager loaded objects, the following can be used:

```
//- Replace "EagerLoadParameters eagerLoadParms = null"
bool useEagerLoading = false
```

```
using SignifyHR.Core;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;

namespace SignifyHR.Data.Domain
{
   public partial class exSample : IAuditable
   {
      #region Search Parameters

      public class SearchParameters : BaseSearchParameters
      {
         public int? SomeId { get; set; }
         public string Description { get; set; }
         public bool IsUsed { get; set; }
      }

      #endregion

      #region Eager Load Parameters

      public class EagerLoadParameters : BaseSearchParameters
      {
         public bool IncludeSampleDocuments { get; set; }
         public bool IncludeSampleComments { get; set; }
      }

      #endregion
```

```
        #region Protected Methods

        protected static IQueryable<exSample> ValidSamples(SignifyHRDAL dbContext, EagerLoadParameters
eagerLoadParms = null)
        {
            var samples = dbContext.exSamples.AsQueryable();

            if (eagerLoadParms != null)
            {
                if (eagerLoadParms.IncludeSampleDocuments )
                    samples = samples.Include(item => item.exSampleDocuments);

                if (eagerLoadParms.IncludeSampleComments )
                    samples = samples.Include(item => item .exSampleComments);
            }

            return samples;
        }

        protected static IQueryable<exSample> FilterSamples(SignifyHRDAL dbContext, SearchParameters
searchParms, EagerLoadParameters eagerLoadParms = null)
        {
            var result = ValidSamples(dbContext, eagerLoadParms);

            if (searchParms!= null)
            {
                if (!String.IsNullOrWhiteSpace(searchParms.Description))
                    result = result.Where(item =>
item.Description.ToLower().Contains(searchParms.Description.ToLower()));

                if (searchParms.SomeId.HasValue)
                    result = result.Where(item => item.SomeId == searchParms.SomeId.Value);

                if (searchParms.IsUsed.HasValue)
                    result = result.Where(item => item.IsUsed == searchParms.IsUsed.Value);
            }

            return result;
        }
```

```
        #endregion

        #region Public Methods

        public static exSample Fetch(ISessionHandler sessionHandler, int id, EagerLoadParameters
eagerLoadParms = null)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                return ValidSamples(dbContext, eagerLoadParms).Single(item => item.Id == id);
            }
        }

        public static exSample TryFetch(ISessionHandler sessionHandler, int id, EagerLoadParameters
eagerLoadParms = null)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                return ValidSamples(dbContext, eagerLoadParms).SingleOrDefault(item => item.Id == id);
            }
        }

        public static IEnumerable<exSample> FetchAll(ISessionHandler sessionHandler, SearchParameters
searchParms, EagerLoadParameters eagerLoadParms = null)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                var results = FilterSamples(dbContext, searchParms, eagerLoadParms);

                return results.OrderByDescending(item => item.Id)
                        .Skip(searchParms.Skip)
                        .Take(searchParms.Take)
                        .ToList();
            }
        }

        #endregion
```

```
        #region Public CRUD Actions

        public void Create(ISessionHandler sessionHandler)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                dbContext.exSamples.Add(this);
                dbContext.SaveChanges();
            }
        }

        public void Update(ISessionHandler sessionHandler)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                dbContext.exSamples.Attach(this);
                dbContext.Entry(this).State = EntityState.Modified;
                dbContext.SaveChanges();
            }
        }

        public void Delete(ISessionHandler sessionHandler)
        {
            using (var dbContext = new SignifyHRDAL(sessionHandler))
            {
                dbContext.exSamples.Attach(this);
                dbContext.exSamples.Remove(this);
                dbContext.SaveChanges();
            }
        }

        #endregion
    }
}
```

Methods

FetchAll - PagedList.IPagedList

OrderBy or *OrderByDescending* **must** be applied prior to *ToPagedList*.

Use *PagedList.IPagedList* for Bootstrap 3 Pager

```
using PagedList;


public static IPagedList<exSample> FetchAll(ISessionHandler sessionHandler, SearchParameters searchParms,
EagerLoadParameters eagerLoadParms = null)
{
  using (var dbContext = new SignifyHRDAL(sessionHandler))
  {
     var results = FilterSamples(dbContext, searchParms, eagerLoadParms);


      return results.OrderByDescending(item => item.Id)
            .ToPagedList(searchParms.CurrentPage.Value, searchParms.PageSize.Value);
  }
}
```

FetchAll - X.PagedList.IPagedList

OrderBy or *OrderByDescending* **must** be applied prior to *ToPagedList*.

Use *X.PagedList.IPagedList* for Bootstrap 3 Pager when returning POCO class data.

```
using X.PagedList;



       public static IPagedList<exSample> FetchAllPreview(ISessionHandler sessionHandler, SearchParameters
searchParms, EagerLoadParameters eagerLoadParms = null)
       {
          using (var dbContext = new SignifyHRDAL(sessionHandler))
          {
             var samples = FilterSamples(dbContext, searchParms, eagerLoadParms);


             var results = samples.Select(item => new SamplePreview
             {
                XSPreviewId = item.Id,
                XSDescription = item.Description,
                XSPath = Path.GetFileNameWithoutExtension(item.Title)
             })
```

```
            return results.OrderByDescending(item => item.Id)
                    .ToPagedList(searchParms.CurrentPage.Value, searchParms.PageSize.Value, results.Count());
        }
    }
```

Create & Update (Non-*IAuditable* Table)

Use the following where table definitions does not contain all columns as required by *IAuditable*.

```
public void Create(exSample sample)
{
    using (var dbContext = new SignifyHRDAL(true))
    {
        sample.SomeValue = 123;

        dbContext.exSample.AddObject(sample);
        dbContext.SaveChanges();
    }
}

public void Update(exSample sample)
{
    using (var dbContext = new SignifyHRDAL(true))
    {
        sample.SomeValue = 123;

        dbContext.exSample.Attach(sample);
        dbContext.ObjectStateManager.ChangeObjectState(this, EntityState.Modified);
        dbContext.SaveChanges();
    }
}
```

Delete Multiple

Use the following for multiple items.

```
public void DeleteMany(IEnumerable<exSample> sampleList)
{
    using (var dbContext = new SignifyHRDAL(true))
```

```
    {
        foreach (var sample in sampleList)
        {
            sample.Delete();
        }
    }
}
```

Meta Data

Remember to specify *MetadataType* for partial class created.

```
[MetadataType(typeof(exSampleMeta))]
public partial class exSample : IAuditable { ... }

public class exSampleMeta : DefaultColumnsMeta
{
    [Required(ErrorMessage = "The {0} field is required")]
    [Display(Name = "Sample Name")]
    public string Description { get; set; }

    [Required(ErrorMessage = "The {0} field is required")]
    [Display(Name = "Start Date")]
    [DataType(DataType.DateTime)]
    public DateTime StartDate { get; set; }

    [Required(ErrorMessage = "The {0} field is required")]
    [Display(Name = "End Date")]
    [DataType(DataType.DateTime)]
    public DateTime EndDate { get; set; }

    [Display(Name = "Enabled")]
    public bool Enabled { get; set; }
}
```

TODO : AddRange, UpdateRange, FetchPaged

LINQ

IQueryable

This is used when you want to add and/or filter results from the database, without the query being executed already. This is generally used with methods like **ValidSamples()** and **FilterSamples()**

The IQueryable interface inherits the IEnumerable interface so that if it represents a query, the results of that query can be enumerated.

IEnumerable

Use when you deal with in process memory object collections and loop through the collection objects. IEnumerable uses Func objects that result in the query being **executed immediately and completely**, your application will see a performance boost.

Example:

```
using SignifyHR.Core;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;

namespace SignifyHR.Data.Domain
{
   public partial class exSample : IAuditable
   {
      #region Search Parameters

      public class SearchParameters : BaseSearchParameters
      {
         public int? SomeId { get; set; }
         public string Description { get; set; }
```

```
        public bool IsUsed { get; set; }
    }

    #endregion

    #region Eager Load Parameters

    public class EagerLoadParameters : BaseSearchParameters
    {
        public bool IncludeSampleDocuments { get; set; }
        public bool IncludeSampleComments { get; set; }
    }

    #endregion

    #region Protected Methods

    protected static IEnumerable<exSample> ValidSamples(SignifyHRDAL dbContext, EagerLoadParameters
eagerLoadParms = null, SearchParameters searchParms)
    {
        //Result not executed yet
        var samples = dbContext.exSamples.AsQueryable();

        //Result not executed yet
        if (eagerLoadParms != null)
        {
            if (eagerLoadParms.IncludeSampleDocuments )
                samples = samples.Include(item => item.exSampleDocuments);

            if (eagerLoadParms.IncludeSampleComments )
                samples = samples.Include(item => item .exSampleComments);
        }

        //Result not executed yet
        if (searchParms!= null)
        {
            if (!String.IsNullOrWhiteSpace(searchParms.Description))
                samples = samples.Where(item =>
item.Description.ToLower().Contains(searchParms.Description.ToLower()));
```

```
            if (searchParms.SomeId.HasValue)
                samples = samples.Where(item => item.SomeId == searchParms.SomeId.Value);

            if (searchParms.IsUsed.HasValue)
                samples = samples.Where(item => item.IsUsed == searchParms.IsUsed.Value);
        }

□□□//Result executed immediately and completely
        return samples.AsEnumerable();
    }
  }
}
```

TODO: Add example for IQueryable

As many calculations, filtering should occur in IQueryable

Naming convention for IQueryable e.g. Valid{ObjectNames}

EF Core

https://docs.microsoft.com/en-us/ef/

JavaScript

Table of contents

General

Try to follow the AirBnB styling guide as far as possible. Sometimes you might have to deviate from the guide; such as when targeting Internet Explorer without a transpiler, you have to use `var` instead of `const` or `let`.

When using a complete toolchain with a transpiler, eslint should be configured to warn about any style violations.

Naming conventions

- Use camelCase when naming objects, functions, and instances.

```
// bad
const OBJEcttsssss = {};
const this_is_my_object = {};
function c() {}

// good
const thisIsMyObject = {};
function thisIsMyFunction() {}
```

- Use PascalCase only when naming constructors or classes.

```
// bad
function user(options) {
  this.name = options.name;
}

const bad = new user({
  name: 'nope',
});

// good
class User {
  constructor(options) {
    this.name = options.name;
  }
}

const good = new User({
  name: 'yup',
});
```

- Acronyms and initialisms should always be all uppercased, or all lowercased.

```
// bad
import SmsContainer from './containers/SmsContainer';

// bad
const HttpRequests = [
  // ...
];

// good
import SMSContainer from './containers/SMSContainer';

// good
const HTTPRequests = [
  // ...
];

// also good
const httpRequests = [
  // ...
];

// best
import TextMessageContainer from './containers/TextMessageContainer';

// best
const requests = [
  // ...
];
```

Bundling

Always use bundling to some degree in production. Commonly used vendor scripts should always be bundles, but however be aware that bundles do not grow too large. Bundling tools such as webpack provide an efficient way to split bundles when they grow too large.

TODO : bundles do not grow too large - what is the guidelines here?

Example of bundling

Minification

Always minify in production.

TODO : Example and how is minification done

Use minified version when external library is used

jQuery

- Prefix jQuery object variables with a `$`.

```
// bad
const sidebar = $('.sidebar');

// good
const $sidebar = $('.sidebar');

// good
const $sidebarBtn = $('.sidebar-btn');
```

- Cache jQuery lookups.

```
// bad
function setSidebar() {
  $('.sidebar').hide();

  // ...

  $('.sidebar').css({
    'background-color': 'pink',
  });
}

// good
function setSidebar() {
  const $sidebar = $('.sidebar');
  $sidebar.hide();

  // ...

  $sidebar.css({
    'background-color': 'pink',
  });
```

```
}
```

- Use `find` with scoped jQuery object queries.

```
// bad
$('ul', '.sidebar').hide();
$('.sidebar').find('ul').hide();
$('.sidebar').children('ul').hide();

// good
$('.sidebar ul').hide();
$('.sidebar > ul').hide();
$sidebar.find('ul').hide();
```

- Handle failure and complete events when performing AJAX calls.

```
$.getJSON('http://example.com/json', (data) => {
  // Do something with the JSON data
}).fail((jqXHR, textStatus, errorThrown) => {
  // Show an error message
}).always(() => {
  // Hide a loading indicator if required
});
```

JavaScript

TypeScript

Iterators (Loops)

- Don’t use iterators. Prefer JavaScript’s higher-order functions instead of loops like `for-in` or `for-of`.
  - Use `map()` / `every()` / `filter()` / `find()` / `findIndex()` / `reduce()` / `some()` / ... to iterate over arrays, and `Object.keys()` / `Object.values()` / `Object.entries()` to produce arrays so you can iterate over objects.

```
const numbers = [1, 2, 3, 4, 5];

// bad
let sum = 0;
for (let num of numbers) {
  sum += num;
}
sum === 15;

// good
let sum = 0;
numbers.forEach((num) => {
  sum += num;
});
sum === 15;

// best (use the functional force)
const sum = numbers.reduce((total, num) => total + num, 0);
sum === 15;

// bad
const increasedByOne = [];
for (let i = 0; i < numbers.length; i++) {
  increasedByOne.push(numbers[i] + 1);
}

// good
const increasedByOne = [];
```

```
numbers.forEach((num) => {
  increasedByOne.push(num + 1);
});

// best (keeping it functional)
const increasedByOne = numbers.map((num) => num + 1);
```

Properties

- Use dot notation when accessing properties.

```
const luke = {
  jedi: true,
  age: 28,
};

// bad
const isJedi = luke['jedi'];

// good
const isJedi = luke.jedi;
```

- Use bracket notation `[]` when accessing properties with a variable.

```
const luke = {
  jedi: true,
  age: 28,
};

function getProp(prop) {
  return luke[prop];
}

const isJedi = getProp('jedi');
```

- Use additional trailing commas for cleaner git diffs.

```
// bad
const hero = {
  firstName: 'Dana',
  lastName: 'Scully'
};
```

```
const heroes = [
  'Batman',
  'Superman'
];

// good
const hero = {
  firstName: 'Dana',
  lastName: 'Scully',
};

const heroes = [
  'Batman',
  'Superman',
];
```

Variables

> Do not use `const` or `let` when targeting Internet Explorer without a transpiler.

- Always use `const` or `let` to declare variables. Not doing so will result in global variables. We want to avoid polluting the global namespace.

```
// bad
superPower = new SuperPower();

// good
const superPower = new SuperPower();
```

- Use `const` for all of your references; avoid using `var`.

```
// bad
var a = 1;
var b = 2;

// good
const a = 1;
const b = 2;
```

- Use one `const` or `let` declaration per variable or assignment.

```
// bad
const items = getItems(),
    goSportsTeam = true,
    dragonball = 'z';

// bad
// (compare to above, and try to spot the mistake)
const items = getItems(),
    goSportsTeam = true;
    dragonball = 'z';
```

```
// good
const items = getItems();
const goSportsTeam = true;
const dragonball = 'z';
```

- Group all your `const`s and then group all your `let`s.

```
// bad
let i, len, dragonball,
    items = getItems(),
    goSportsTeam = true;

// bad
let i;
const items = getItems();
let dragonball;
const goSportsTeam = true;
let len;

// good
const goSportsTeam = true;
const items = getItems();
let dragonball;
let i;
let length;
```

- Assign variables where you need them, but place them in a reasonable place.

```
// bad - unnecessary function call
function checkName(hasName) {
  const name = getName();

  if (hasName === 'test') {
    return false;
  }

  if (name === 'test') {
    this.setName('');
    return false;
  }
```

```
  return name;
}

// good
function checkName(hasName) {
  if (hasName === 'test') {
    return false;
  }

  const name = getName();

  if (name === 'test') {
    this.setName('');
    return false;
  }

  return name;
}
```

- Don’t chain variable assignments.

```
// bad
(function example() {
  // JavaScript interprets this as
  // let a = ( b = ( c = 1 ) );
  // The let keyword only applies to variable a; variables b and c become
  // global variables.
  let a = b = c = 1;
}());

console.log(a); // throws ReferenceError
console.log(b); // 1
console.log(c); // 1

// good
(function example() {
  let a = 1;
  let b = a;
  let c = a;
```

```
}());

console.log(a); // throws ReferenceError
console.log(b); // throws ReferenceError
console.log(c); // throws ReferenceError

// the same applies for `const`
```

- Avoid linebreaks before or after `=` in an assignment. If your assignment violates `max-len`, surround the value in parens

```
// bad
const foo =
  superLongLongLongLongLongLongLongFunctionName();

// bad
const foo
  = 'superLongLongLongLongLongLongLongString';

// good
const foo = (
  superLongLongLongLongLongLongLongFunctionName()
);

// good
const foo = 'superLongLongLongLongLongLongLongString';
```

CSS/HTML

Table of contents

General

Try to follow the AirBnB CSS/SASS guideline as far as possible.

Please have a look at the rest of this chapter to see more details about Naming Conventions and **BEM (very important!)**

CSS/HTML

Naming conventions

BEM, or “Block-Element-Modifier”, is a *naming convention* for classes in HTML and CSS.

We will be using BEM for the following reasons:

- It helps create clear, strict relationships between CSS and HTML
- It helps us create reusable, composable components
- It allows for less nesting and lower specificity
- It helps in building scalable stylesheets

Example:

HTML

```
<a class="btn btn--big btn--orange" href="https://css-tricks.com">
  <span class="btn__price">R199.99</span>
  <span class="btn__text">Subscribe</span>
</a>
```

CSS

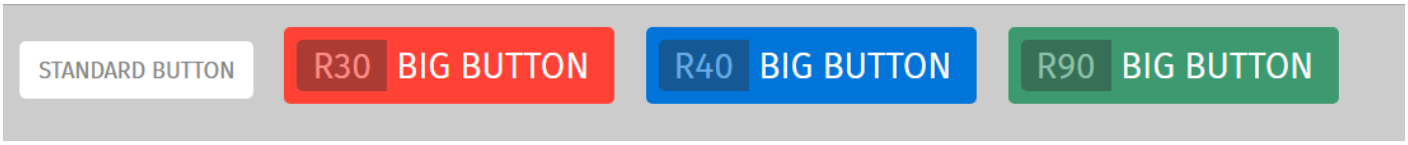
```
/* Block component */
.btn {}

/* Element that depends upon the block */
.btn__price {}

/* Modifier that changes the style of the block */
.btn--orange {}
.btn--big {}
```

- In this CSS methodology a **block** is a top-level abstraction of a new component, for example a button: `.btn { }`. This block should be thought of as a parent.
- Child items, or **elements**, can be placed inside and these are denoted by two underscores following the name of the block like `.btn__price { }`.
- Finally, **modifiers** can manipulate the block so that we can theme or style that particular component without inflicting changes on a completely unrelated module. This is done by appending two hyphens to the name of the block just like `btn--orange`.

If another developer wrote this markup, and we weren't familiar with the CSS, we should still have a good idea of which classes are responsible for what and how they depend on one another. Developers can then build their own components and modify the existing block to their heart's content. Without writing much CSS, developers are potentially capable of creating many different combinations of buttons simply by changing a class in the markup:



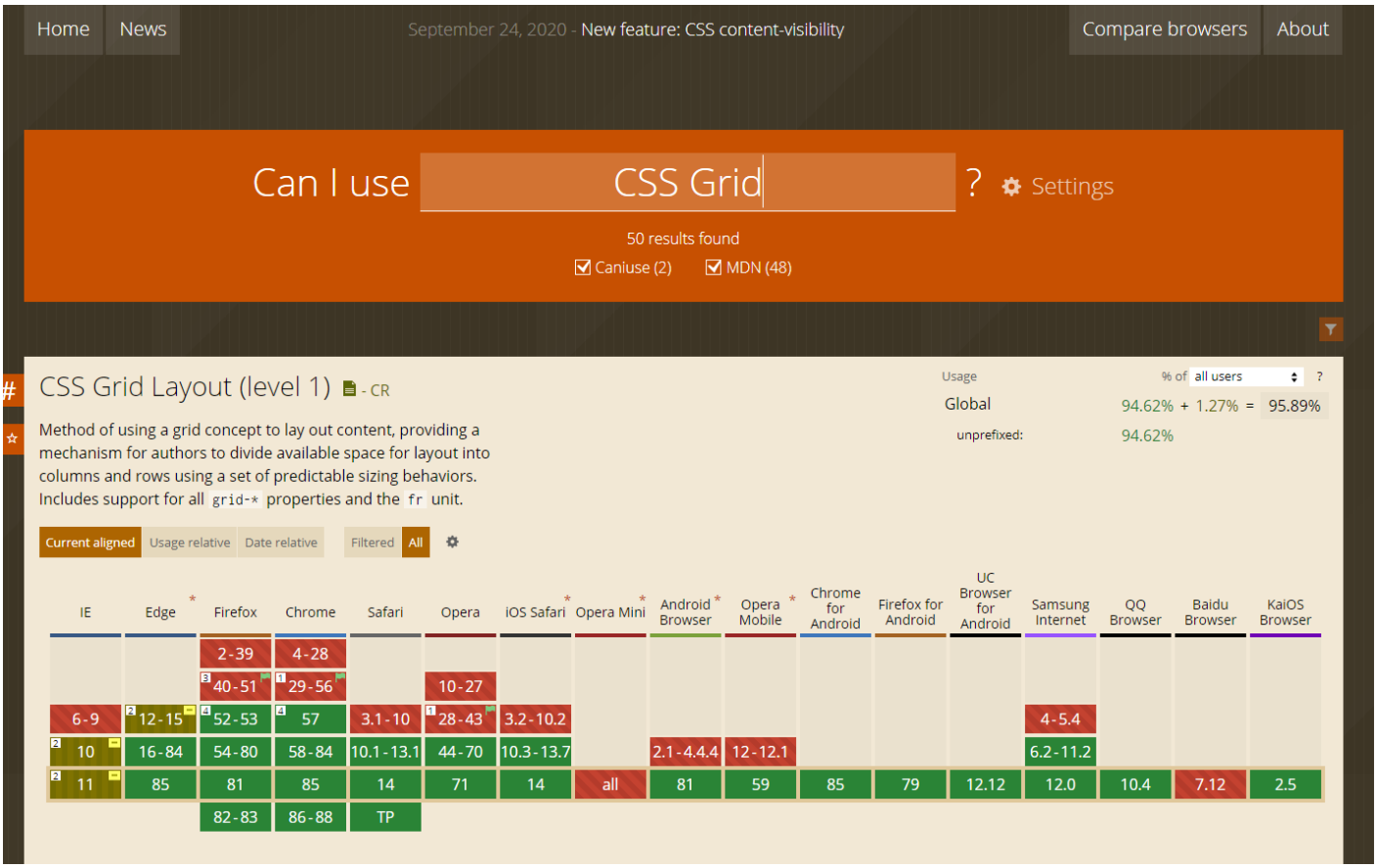
CSS/HTML

Can I Use

To make sure the supported browsers can render the CSS properties that you want to use, refer to the Can I Use Site.

Can I Use provides browser support tables for modern web technologies.

For example, when searching CSS Grid, you will see all the browsers (and their versions) that can support the properties:



TO DO : show list of browsers supported by Signify

Bundling

Always use bundling to some degree in production. Commonly used CSS files (for an Area or Module) should always be bundled.

TODO : Give more info and examples on CSS bundling

CSS/HTML

SASS

Sass is a stylesheet language that’s compiled to CSS. It allows you to use variables, nested rules, mixins, functions, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized and makes it easy to share design within and across projects.

- Click **here** to view the documentation for SASS

Naming Conventions

Style rules are the foundation of Sass, just like they are for CSS. And they work the same way: you choose which elements to style with a selector, and declare properties that affect how those elements look.

Nested selectors

Do not nest selectors more than three levels deep!

```
.page-container {
  .content {
    .profile {
      // STOP!
    }
  }
}
```

When selectors become this long, you're likely writing CSS that is:

- Strongly coupled to the HTML (fragile) —*OR*—
- Overly specific (powerful) —*OR*—
- Not reusable

Again: **never nest ID selectors!**

If you must use an ID selector in the first place (and you should really try not to), they should never be nested. If you find yourself doing this, you need to revisit your markup, or figure out why such strong specificity is needed. If you are writing well formed HTML and CSS, you should **never** need to do this.

>

> Please refer to SASS Style Rules for more details

Variables

Sass variables are simple: you assign a value to a name that begins with `$`, and then you can refer to that name instead of the value itself. But despite their simplicity, they're one of the most useful tools Sass brings to the table. Variables make it possible to reduce repetition, do complex math, configure libraries, and much more.

Prefer dash-cased variable names (e.g. `$my-variable`) over camelCased or snake_cased variable names. It is acceptable to prefix variable names that are intended to be used only within the same file with an underscore (e.g. `$_my-variable`).

> Please refer to the Variables for examples and details

Mixins

Mixins allow you to define styles that can be re-used throughout your stylesheet. They make it easy to avoid using non-semantic classes like `.float-left`, and to distribute collections of styles in libraries.

Mixins should be used to DRY up your code, add clarity, or abstract complexity--in much the same way as well-named functions. Mixins that accept no arguments can be useful for this, but note that if you are not compressing your payload (e.g. gzip), this may contribute to unnecessary code duplication in the resulting styles.

> Please refer to the @mixin and @include for examples and details

Functions

Sass provides many built-in modules which contain useful functions (and the occasional mixin). These modules can be loaded with the `@use` rule like any user-defined stylesheet, and their functions can be called like any other module member. All built-in module URLs begin with `sass:` to indicate that they're part of Sass itself.

> Please refer to the Functions documentation for examples and details

Ordering of property declarations

1. Property declarations

List all standard property declarations, anything that isn't an `@include` or a nested selector.

```
.btn-green {
  background: green;
  font-weight: bold;
  // ...
}
```

2. `@include` declarations

Grouping `@include`s at the end makes it easier to read the entire selector.

```
.btn-green {
  background: green;
  font-weight: bold;
  @include transition(background 0.5s ease);
  // ...
}
```

3. Nested selectors

Nested selectors, *if necessary*, go last, and nothing goes after them. Add whitespace between your rule declarations and nested selectors, as well as between adjacent nested selectors. Apply the same guidelines as above to your nested selectors.

```
.btn {
  background: green;
  font-weight: bold;
  @include transition(background 0.5s ease);

  .icon {
    margin-right: 10px;
  }
}
```

React

Design Principles and Standards

For design principles using React, use the guide below by clicking the image

For styling using React and JSX, use the guide below by clicking the image

Web API (REST)

Table of contents

General

This is an extension on C# standards.

XML Documentation Commenting

Code Documenting

Ensure all publicly exposed actions, objects and properties are described using XML Documentation Comments as this is used to instruct third party users more about what an action does, what an object represents and what a property entails.

```
/// <summary>
/// Returns a sync structure list of all first level nodes to which user has access
/// </summary>
/// <returns></returns>
[ApiRoute("LearningPortal/AllCourses", AcceptedVersions = new[] { 1, 2, 3 })]
public IEnumerable<NodeSyncStructure> GetNodes()
{
    try
    {
        return NodeSyncStructure.FetchAll(SessionHandler, false);
    }
    catch (Exception ex)
    {
        throw LogException(HttpStatusCode.BadRequest, Request, ex);
    }
}
```

```
/// <summary>
/// Data Tranfer Object to transfer node comparison information from and to the API
/// </summary>
public class NodeSyncStructure
{
    /// <summary>
    /// The id for the node
    /// </summary>
```

```
    public int NodeId { get; set; }
    /// <summary>
    /// The latest edited date for the node - server date
    /// </summary>
    public DateTime NodeLastEditedDate { get; set; }
    /// <summary>
    /// The latest edited date for the employee node progress - server date
    /// </summary>
    public DateTime EmployeeProgressLastEditedDate { get; set; }
    /// <summary>
    /// The number of user who completed the node
    /// </summary>
    public int CompletedCount { get; set; }
}
```

GET /api/v{version}/LearningPortal/AllCourses — Returns a sync structure list of all first level nodes to which user has access

Response Class (Status 200)

Model | Model Schema

Inline Model [
SignifyHR.API.Areas.Learning.DTO.NodeSyncStructure
]
SignifyHR.API.Areas.Learning.DTO.NodeSyncStructure {
NodeId (integer, *optional*): The id for the node,
NodeLastEditedDate (string, *optional*): The latest edited date for the node - server date,
EmployeeProgressLastEditedDate (string, *optional*): The latest edited date for the employee node progress - server date,
CompletedCount (integer, *optional*): The number of user who completed the node,
OverallRating (number, *optional*): The overall rating for node
}

Naming conventions

URI's

API URI's are based around resources. Actions are indicated with the use of HTTP Methods.

- Use nouns, not verbs
- Resources that are collections should be indicated using plural nouns
  e.g. `/pathways`
- Specific resource in collection is defined after collection (generally with *id*)
  e.g. `/pathways/{id}`
- Resources of which only one instance can exist should be indicated using singular noun
  e.g. `/pathways/{id}/favourite`
- Related/Linked resources are defined after specific resource (recursive - refer to above)
  e.g. `/pathways/{nodeId}/steps`, `/pathways/{nodeId}/steps/{stepId}`, `/pathways/{nodeId}/steps/{stepId}/progress`

Microsoft Documentation: Organize the API around resources.

Code

Decorate actions and controllers according to resource path and applicable version(s).

Controllers

```
[ApiRoute("TrainingRequirements")]
public class TrainingRequirementsController : ApiBaseController
{
    //code logic
}
```

Actions

```
[ApiRoute("LearningPortal/Pathways/{nodeId:int}")]
public NodeStructure GetPathway(int nodeId)
{
    //code logic
```

```
}


[ApiRoute("LearningPortal/Pathways/{nodeId:int}/Favourite")]
public bool GetFavourite(int nodeId)
{
   //code logic
}

[ApiRoute("LearningPortal/Pathways/{nodeId:int}/Steps/{stepId:int}", AcceptedVersions = new[] { 1, 2 })]
public StepStructure GetPathwayStep(int nodeId, int stepId)
{
   //code logic
}

[ApiRoute("LearningPortal/Pathways/{nodeId:int}/Steps/{stepId:int}", StartingVersion = 3)]
public StepStructureV3 GetPathwayStepV3(int nodeId, int stepId)
{
   //code logic
}
```

Methods

Related Documentation

Define operations in terms of HTTP methods

Conform to HTTP semantics

GET methods

Action: Retrieve resource(s).
HTTP Status Code(s): 200 (OK), 404 (Not Found)

POST methods

Action: Create resource(s) (can also be used for updates in some cases).
HTTP Status Code(s): 201 (Created), 204 (No Content), 400 (Bad Request), 200 (OK)

PUT methods

Action: Update resource(s).
HTTP Status Code(s): 200 (OK), 204 (No Content), 404 (Not Found), 409 (Conflict)

DELETE methods

Action: Delete/Remove resource(s).
HTTP Status Code(s): 204 (No Content), 404 (Not Found)

Versioning

URI versioning is utilised. Refer to below links for more information:

Versioning a RESTful web API

API Version Control

Service Versioning

Object Usage

Data Transfer Object - DTO

Data Transfer Object is the object returned by an API call.
A DTO is used to prevent exposing your database entity structure. This also prevent the exposing of data not being or to be used by the API client, such as linked entities. Data can also be returned as human readable if an API call is consumed by an UI client.

```
/// <summary>
/// Data Tranfer Object to transfer employee node rating information from and to the API
/// </summary>
public class EmployeeNodeRating
{
    /// <summary>
    /// The id of the rating
    /// </summary>
    public int Id { get; set; }
    /// <summary>
    /// The node id for pathway rated
    /// </summary>
    public int NodeId { get; set; }
    /// <summary>
    /// The rating value specified
    /// </summary>
    public int Rating { get; set; }
    /// <summary>
    /// The comment specified for ranking
    /// </summary>
    public string Comment { get; set; }
    /// <summary>
    /// The employee name for who rated node
    /// </summary>
    public string EmployeeName { get; set; }
    /// <summary>
    /// The rating date for ranking
```

```
    /// </summary>
    public string Date { get; set; }
    /// <summary>
    /// The image url for user rating
    /// </summary>
    public string ImageUrl { get; set; }
    /// <summary>
    /// The flag to indicate if comment is own comment
    /// </summary>
    public bool IsOwnComment { get; set; }

    /// <summary>
    /// Fetch the node rating details
    /// </summary>
    /// <param name="sessionHandler"></param>
    /// <param name="nodeId"></param>
    /// <returns></returns>
    public static EmployeeNodeRating Fetch(ISessionHandler sessionHandler, int nodeId)
    {
        var eagerLoadParms = new pwEmployeeRating.EagerLoadParameters
        {
            IncludeEmployees = true,
        };

        var empRating = pwEmployeeRating.TryFetchByNodeAndEmployee(sessionHandler, nodeId,
sessionHandler.EmployeeId.Value, eagerLoadParms);

        if (empRating == null)
            return null;
        else
            return new EmployeeNodeRating
            {
                Id = empRating.Id,
                NodeId = empRating.NodeId,
                EmployeeName = empRating.prsEmployee.NameSurname,
                Comment = empRating.Comment,
                Date = empRating.CreatedDate.ToShortDateString(),
                ImageUrl = String.Format("~/api/thumbnail/AspectRatioEmployee?id={0}&width=null&height=96",
empRating.prsEmployee.EmployeeNumber),
                IsOwnComment = true,
```

```
            Rating = empRating.Rating
        };
    }
}
```

Value/View Object - VO

A Value/View Object is generally the object received by an API call.
This is similar to a DTO in the sense that the underlying database entity is not exposed. A VO generally only contains properties for all the applicable, editable values to be submitted by the client; validation are also performed on these values prior to submitting data to data storage / database.

```
/// <summary>
/// View Object to transfer employee node rating information from the API
/// </summary>
public class EmployeeNodeRating
{
    /// <summary>
    /// The id for node
    /// </summary>
    public int NodeId { get; set; }
    /// <summary>
    /// The rating value specified
    /// </summary>
    public int Rating { get; set; }
    /// <summary>
    /// The comment specified for ranking
    /// </summary>
    public string Comment { get; set; }
    /// <summary>
    /// The flag to exclude comment from updating
    /// </summary>
    public bool IgnoreCommentChange { get; set; }

    /// <summary>
    /// Log the node rating for employee
    /// </summary>
    /// <param name="sessionHandler"></param>
```

```
    /// <returns></returns>
    public void LogRating(ISessionHandler sessionHandler)
    {
        var empRating = pwEmployeeRating.TryFetchByNodeAndEmployee(sessionHandler, this.NodeId,
sessionHandler.EmployeeId.Value);
        if (empRating == null)
        {
            empRating = new pwEmployeeRating
            {
                NodeId = this.NodeId,
                EmployeeId = sessionHandler.EmployeeId.Value,
                Rating = this.Rating,
                Comment = this.IgnoreCommentChange ? String.Empty : this.Comment.Trim(),
            };

            empRating.Create(sessionHandler);
        }
        else
        {
            empRating.Rating = this.Rating;
            empRating.Comment = this.IgnoreCommentChange ? empRating.Comment : this.Comment.Trim();
            empRating.Update(sessionHandler);
        }
    }
}
```

SQL

Table of contents

General

- No more than 200 lines (formatted) allowed in any stored procedure, user-defined function or view. *If more than 200 lines is unavoidable, please discuss this with a Senior Developer or Database Administrator first.*
- Aliases should be used for tables in SELECTs for readability purposes and alias names should make sense. If alias is too long (>10 characters) , use an abbreviation that makes sense.
- **EditedUser** in a SQL batch job is the name of the stored procedure where it is called from. When the edited user is provide to the procedure the SP name must be appended to e.g. {Username}_MyProc. This must be applied to any parent procedure that perform data modification.
- Script name assigned correctly http://shakespeare/MasterBuilder/Tools/GenerateScript when committing your scripts.
- *Stored Procedure, Views and Functions heading convention* followed (see example below)

> 2018/11/30 : dbotha : 56789 : Added check for qualifications offered by my learning provider.
> {Date} : {Author} : {TP#} Short concise description of change

- **UPDATE** statements should always contain a **WHERE** clause.
- **WITH(NOLOCK)** should be used in SELECTs to prevent unnecessary locking.
- **SQL KEYWORDS** should always be in **CAPS.**
- Dynamic SQL should only be used when absolutely necessary.
- The **SchemaId** column in tables must not be nullable.
- The statement **SET NOCOUNT ON** should be at the top of the stored procedure unless in the unlikely case where the counts obtained from the stored procedure is used in code.
- When performing an INSERT, always specify the column list

  > **Correct** : INSERT INTO tmp (Value) SELECT @variable INSERT INTO tmp (Value) VALUES(@variable)
  >
  > **Not correct** : INSERT INTO tmp SELECT @variable INSERT INTO tmp VALUES(@variable)

- When writing an automation stored procedure in SQL and the code becomes too complex and long-winded, break the stored procedure into multiple stored procedures.
- Using in-line user defined functions in SELECT statements should only be used if absolutely necessary. NOTE : Complex user-defined functions used in-line in large SELECT statements returning many rows (1000+) can potentially slow down your code

significantly. In-line user defined functions should be tested on large amounts of data.

SQL

Naming conventions

General

- Decide per module if abbreviation (e.g. *prs* for Personnel module) or full name (e.g. *leave*) will be used for database Objects.
- Do not use spaces in the names of database objects.
- Avoid using **ntext**, **text**, and **image** data types in new development work. Use nvarchar (max), varchar (max), and varbinary (max) instead.

Note: The parent / grouping determines the module the data is stored on E.g. EmployeePDPs (an employee's PDP's) vs. pdpPeriod (a PDP's periods)

Table 1 : Database Module Abbreviations

Abbreviation	**Module**
cc	Career Conversation
cfg	Configuration
cl	Catalogue
com	Communication (Import / Export)
cpd	Credits
dbd	Dashboards
ab	Assessment Builder
ee	Employee Evaluation
el2	E-Learning v2
el	e-Learning

els	Learning Store
em	Event Management / Training and Scheduling
fais	FAIS
ate	Ask The Expert/ Discussion Forum
icn	Icodeon
ir	Internal Relations / Disciplinary Actions
jl	Job Leveling
jp	Job Profiler
leave	Leave Management - New
lic	Licences
mc	Mentors and Coaches
ntf	Notifications
org	Organisational Structure
pdm	Performance Management
pdp	Personal Development Plan
prc	HR Processes
prs	Personnel
pw	Pathways
rb	Report Builder
rec	Recruitment

rem	Remuneration
rp	Resource Planning
rpt	Reports / Report Management
sms	SMS Notifications
sr	Salary Review
ss	Salary Scenario
sty	System Framework
sys	System Administration
tal	Talent Management - New
sc	Succession and Career Planning
tM	Talent Management - Old
txAudit	Auditing - Old
wf	Work Flows
tr_	Trigger

Tables

- A table name must always be prefixed with the module name abbreviation (see above).
- A database table name must always be plural
  - *prsEmployees* – There will most likely be more than one employee in the system
  - *LeaveGroupTypes* – Each Leave Group can have one or more Leave Type
  - *eeImports* – Only one import can run at a time
  - *pdpStatuses* – There are multiple statuses for the module
- A column name must be the shortest descriptive name possible
  - Do not specify module prefix e.g.

> ***Correct column name*** *EmployeeId*

> ***Incorrect column name*** *prsEmployeeId*

    - **Exceptional case:** If more than one column in the same table are the “same” e.g. *CategoryId*, specify module prefix e.g. *cpdCategoryId, pdpCategoryId*
- A column name must refer to a single and not multiple instances
    - Use *UnitId* instead of *UnitsId*
- Rather use **varchar(max)** instead of **text** or **varchar(8000)** types for string columns where applicable
- Each table that has a single identity column must also have a clustered primary key with the following naming convention:
    - PK_{*TableName*}_{*IdentityColumn*}
- A foreign key constraint name must be in the following format:
    - FK_{*TableName*}_{*Column1*}

Views

- A View’s name must follow the same convention as table names (add *View* at end of name)
    - Use *prsTerminatedEmployeesView* instead of *viewTerminatedEmployees*
    - *naming: {prefix}{Description of the data resturned}View*

User-Defined Functions

- A User-Defined function's name must follow the same convention as table names.
- A user defined function must be prefixed with the module (do not add *fnc* prefix)
    - Use *el2SelectScholarshipManagerNotificationDays* instead of *fncSelectScholarshipManagerNotificationDays*

- General user defined functions (module-unspecific) can be the description of the output
    - Use *Split* instead of *fncSplit*
    - Use *CleanHtmlTags* instead of *fncCleanHTMLTags*

Stored Procedures

- A stored procedure name must always be prefixed with the module.
- A stored procedure must indicate it’s intention by using a keyword on what action will be performed
    - Select
    - InsertUpdate
    - Insert
    - Update
    - Delete
    - Check/ Verify
    - Copy
    - Archive
    - Reset
    - Apply
- Examples

- prsSelectEmployeesAll
- prsSelectEmployeesList - paging
- prsSelectEmployee - single
- prsInsertUpdateEmployee
- prsDeleteEmployee
- prsCheckEmployeeIDNumber
- styResetUserPassword
- pdmCopyContract

Temporary Tables

- Single use temp table: #{Descriptive table name}
- Global use temp table: ##{Descriptive table name}
- Variable temp table @{Descriptive table name}

Common Table Expressions (CTE's)

- CTE table names are declared with the prefix **cte.**

Indexes

- A non-clustered index name must be in the following format:
  - IX_{*TableName*}_{*Column1*}_{*Column2*}
  - Indexes have a maximum size of 900 or 1700 depending on the index type and SQL version. Do not create a non-clustered index on a column with a max length of more than 500.

- Always check with the Database Administrator whether indexes should be created during development. Assume that indexes will always be created.

Constraints

- A default constraint name must be in the following format:
  - DF_{*TableName*}_{*Column1*}

- A unique constraint name must be in the following format:
  - UQ_{*TableName*}_{*Column1*}_{*Column2*}

- A check constraint name must be in the following format:
  - CK_{*TableName*}_{*Column1*}_{*Column2*}

- Columns with Default value constraint should not allow NULLs.

TODO: Add a link to main page for each section

Cursors

- Use cursors only when absolutely necessary.
- If the function performed by the cursor could have been achieved by another SQL function e.g. PIVOT or Common Table Expression then rather do that as CURSORS are expensive.
- When using a cursor to only cycle once through records without updating them, use the following syntax to make the cursor as light as possible:

```
DECLARE @SchemaID INT

DECLARE curs CURSOR LOCAL FORWARD_ONLY STATIC READ_ONLY FOR
  SELECT
    SchemaID
  FROM cfgSchemaID WITH(NOLOCK)
  WHERE
    GETDATE() BETWEEN ValidFrom AND ValidTo
    AND SysID = 101

OPEN curs

FETCH NEXT FROM curs
INTO @SchemaID

WHILE @@FETCH_STATUS = 0
BEGIN

     /*Do your commands for @SchemaID here*/

/*Get the next author.*/
FETCH NEXT FROM curs
INTO @SchemaID
END

CLOSE curs
DEALLOCATE curs
```

- When evaluating the use of cursors first consider the use of for XML path to loop through each item in a table

SQL

Common table expressions (CTE's)

- CTE table names are declared with the prefix **cte**
- Used to simplify complex joins and subqueries.
- Use a Common Table Expression for paging instead of Dynamic SQL.
- Always start with a semi-colon before the WITH.
- Chaining CTE's must be limited to 3 instances.
- CTE's must be filtered as soon as possible to limit the number of records stored in memory.
- CTEs can only be used when data is only required for a single use in the procedure.
- CTEs must always be provided named column and not use the * selector.

```
;WITH cteEmployees
    AS (SELECT
              Name
            , Surname
            , EmployeeNumber
        FROM
            prsEmployees WITH(NOLOCK))
    SELECT
          *
    FROM
        cteEmployees
```

The use of recursive CTEs

- Always ensure a termination condition is defined.
- For an example view this site
- e.g.

```
WITH Managers AS
(
--initialization
SELECT EmployeeID, LastName, ReportsTo
```

```
FROM Employees
WHERE ReportsTo IS NULL
UNION ALL
--recursive execution
SELECT e.employeeID,e.LastName, e.ReportsTo
FROM Employees e INNER JOIN Managers m
ON e.ReportsTo = m.employeeID
)
SELECT * FROM Managers
```

SQL

Temporary tables

- Temp tables are used for the large temporary storage of data.
- Only use local temp tables.
- Use temporary tables cautiously / only when necessary e.g. early filtering in reports / complex queries.
- When a temporary table is used in a stored procedure, evaluate if it is absolutely necessary.
- Ensure that temporary table are always explicitly dropped at the end of the stored procedure.
- When the possibility exist that the temp table does not exist test its existence in the temp..DB before dropping e.g. conditionally created temp table

```
IF OBJECT_ID('tempdb..#TheTable') IS NOT NULL
BEGIN
/*Do Stuff*/
END
```

- Create the table before addition when a fixed definition is required or multiple data sources are used to populate it

```
    CREATE TABLE #LeaveCycles(
        StartDate   DATETIME,
        EndDate     DATETIME,
        LeaveTypeId INT,
        Name        VARCHAR(200) COLLATE DATABASE_DEFAULT,
        CycleId     INT,
        CreatedDate DATETIME,
        EmployeeId  INT
                    )

    INSERT INTO #LeaveCycles(
          StartDate
        , EndDate
        , LeaveTypeId
        , Name
        , CycleId
```

```
        , CreatedDate
        , EmployeeId
                )
    EXEC LeaveCalculateActiveCyclesByLeaveType
        @EmployeeId = @EmployeeId,
        @SchemaId = @SchemaId,
        @LeaveTypeId = @LeaveTypeId,
        @IsHistoric = @IsHistoric
```

- When a single table is used a select into can be done to create the temp table

SQL

Table variables

- Use table variables over temp tables for a small quantity of data (thousands of bytes)

https://stackoverflow.com/questions/11857789/when-should-i-use-a-table-variable-vs-temporary-table-in-sql-server

- Example:

```
DECLARE @product_table TABLE (
    product_name VARCHAR(MAX) NOT NULL,
    brand_id INT NOT NULL,
    list_price DEC(11,2) NOT NULL
);

INSERT INTO @product_table
(
    product_name,
    brand_id,
    list_price
  )
SELECT
    product_name,
    brand_id,
    list_price
FROM
    production.products
WHERE
    category_id = 1;
```

Tables and indexes

- Always use a column list in INSERT statements of SQL queries. This will avoid a problem when table structure changes.

  > **Correct** : INSERT INTO tmp (Value) SELECT @variable
  >
  > INSERT INTO tmp (Value) VALUES(@variable)

  > **Not correct** : INSERT INTO tmp SELECT @variable
  >
  > INSERT INTO tmp VALUES(@variable)

- Perform all referential integrity checks and data validations using constraints instead of triggers, as they are faster.
- Remember to add foreign-key constraints where a table references another.
- Always check with the Database Administrator to confirm what indexes should be added when a new table is added to the database.

SQL

User defined functions

- Do not call functions repeatedly in stored procedures, triggers, functions and batches, instead call the function once and store the result in a variable, for later use.
- Unless absolutely necessary, DO NOT USE in-line user-defined functions in SELECTs. *If unavoidable, discuss with a Senior Developer first before implementing it.*
- *When used it must be if possible only use data provided and not do extra selects from other tables.*

SQL

Stored procedures

- **EXCEPT** or **NOT EXIST** clause can be used in place of LEFT JOIN or NOT IN for better performance (see example for **EXCEPT** below)

```
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salery > 1000
EXCEPT
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salery > 2000
ORDER BY EmpName;
```

- If stored procedure always returns single row resultset, then consider returning the resultset using OUTPUT parameters instead of SELECT statement
- Use query hints to prevent locking if possible (NoLock)
- Avoid using dynamic SQL statements if you can write T-SQL code without using them.
- The number of nested procedures must be limited to no more than 32

SQL

Triggers

- Limit the use of triggers only for auditing, custom tasks, and validations that cannot be performed using constraints.
- If possible only exec trigger conditionally e.g. modifying data

SQL

Existence checks

- Make use of the existence checking defined here
- Always check for existence when adding new objects to the database (see example below)

```
IF NOT EXISTS
        (
        SELECT TOP 1
            1
        FROM
            sys.all_columns c
            JOIN sys.tables t
              ON t.object_id = c.object_id
        WHERE t.name = 'EmployeeLeave'
             AND c.name = 'ActionStatus')
  BEGIN
    ALTER TABLE EmployeeLeave
    ADD
          ActionStatus INT
END
```

- Also check for existence when editing a database object (see example below)

```
IF EXISTS
        (
        SELECT TOP 1
            1
        FROM
            sys.all_columns c
            JOIN sys.tables t
              ON t.object_id = c.object_id
        WHERE t.name = 'EmployeeLeave'
             AND c.name = 'ActionStatus')
  BEGIN
    ALTER TABLE EmployeeLeave
```

```
    ALTER COLUMN ActionStatus NVARCHAR(2) NOT NULL
END
```

SQL

Views

- Incorporate your frequently required, complicated joins and calculations into a view so that you don’t have to repeat those joins/calculations in all your queries. Instead, just select from the view.
- In views always define selects with named columns.
- Avoid the use of views within views.
- If possible rather implement procedures to get filtered datasets.

Supported Browsers

Version 9 Supported Browsers

Signify Supported Browsers			
Desktop Browsers		**Mobile Browsers**	
Chrome	Last 2 browser versions	Chrome	Last 2 browser versions
Mozilla Firefox	Last 2 browser versions	Mozilla Firefox	Last 2 browser versions
Edge	Version 12+	Edge	Version 12+
Safari	Version 10+	Safari	Version 10+
Opera	Last 2 browser versions	Opera	Last 2 browser versions
iOS	Version 10+	iOS	Version 10+