Sitecore CMS 6.5 or later

Sitecore Pipelines

A description of important pipelines in Sitecore CMS

Table of Contents

Chapter 1

Introduction

This document describes the Sitecore pipelines and focuses on the core pipelines and the DMS processors and pipelines.

This manual contains the following chapters:

- Core Pipelines
- DMS Pipelines

1.1 Pipelines

Pipelines define a sequence of processors that implement a function, such as defining the Sitecore context for an HTTP request or generating a list of messages in the Content Editor. Pipelines assist with encapsulation, flexible configuration, separation of concerns, testability, and other objectives.

Each processor in a pipeline contains a method named `Process()` that accepts a single argument and returns void. A processor can abort the pipeline, preventing Sitecore from invoking subsequent processors.

The argument that is passed to the `Process()` method must be of a type that is specific to the pipeline or be the default argument — `Sitecore.Pipelines.PipelineArgs`. To create a pipeline processor, create a class that implements a method named `Process()` with the same signature as the other processors in the pipeline. This processor can inherit from an existing processor, and you can add, remove, replace, and rearrange processors in the pipelines to suit your requirements.

You can use the configuration factory to define parameters for pipeline processors. For more information about the configuration factory, see the blog post The Sitecore ASP.NET CMS Configuration Factory.

Sitecore separates the pipelines into two groups. These groups are defined in the `web.config` file in the `/configuration/sitecore/pipelines` and `/configuration/sitecore/processors` elements .

In general, the pipelines defined in the `/configuration/sitecore/pipelines` element tend to define system processes. They are sometimes referred to as the core pipelines.

The pipelines defined in the `/configuration/sitecore/processors` element operate for UI requests and can interact with the user. They are sometimes referred to as UI pipelines.

Some Sitecore modules introduce additional pipelines through the `Web.config` include files. For more information about web.config include files, see the blog post All About web config Include Files with the Sitecore ASP.NET CMS.

Chapter 2

Core Pipelines

This chapter describes the pipelines defined within the `/configuration/sitecore/pipelines` element in the `web.config` file. These pipelines tend to define system processes and are sometimes referred to as core pipelines.

This chapter contains the following sections:

- HttpBeginRequest Pipeline

2.1 HttpBeginRequest Pipeline

The processors in the `HttpBeginRequest` pipeline generate and populate the `Sitecore.Context` class.

The `httpRequestBegin` pipeline is invoked in the `HttpApplication.BeginRequest` event handler, which is defined in the `Sitecore.Nexus` module. The Processors in the `httpRequestBegin` pipeline can prevent Sitecore from further processing the request, returning control to ASP.NET, for example, if the URL matches the `IgnoreUrlPrefixes` setting in the `Web.config` file.

The `Sitecore.Context` class contains information about the current HTTP request, for instance:

- The Context item.
- The Context site.
- The Context database.
- The Context language.
- The Context device.
- The Context domain.
- The Context user.
- The Context raw URL.
- The Context page mode.

All the processors in the `httpRequestBegin` pipeline inherit from the `HttpRequestProcessor` class and take the `HttpRequestArgs` arguments.

This pipeline contains the following processors:

CheckIgnoreFlag

This processor may abort the `HttpBeginRequest` pipeline depending on the `ignoreFlagName` property. This property is determined by the `FilterUrlExtensions` processor.

StartMeasurements

This processor records statistical measurements, such as the number of items that are accessed during this request, the total amount of memory required for this request, and the raw URL of this request.

IgnoreList

This processor determines whether certain URL prefixes should be processed as friendly URLs or not. The processor uses the `IgnoreUrlPrefixes` setting in `Web.config`.

SiteResolver

This processor selects one of the sites from the `<sites>` section in `Web.config` and sets it as the context site. The processor resolves the site name in one of the following ways:

- By the `sc_site` parameter in the URL.
- By the host name or virtual folder name.

For information about configuring Sitecore sites, see the article *Configuring Multiple Sites* on the SDN.

UserResolver

This processor checks whether the `AuthenticationManager.GetActiveUser` method returns null. If it does, the processor throws an exception and aborts the pipeline.

If you want to customize the `UserResolver` class to set your custom user as the context user you need to call the `AuthenticationManager.SetActiveUser` method in your custom processor.

When you are in Preview mode, this processor uses the user that is stored in the `sc_pview_shuser` cookie to set the current user.

DatabaseResolver

This processor uses the `sc_database` query string parameter if it is specified in the URL to set the context database.

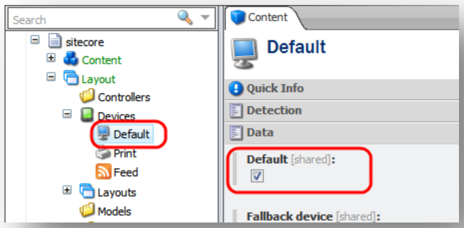
If this parameter is not specified, Sitecore uses the `<site>` setting in `Web.config` to set the context database.

DeviceResolver

This processor sets the context device. If the context device has already been set, the processor has no effect.

If the device is not set yet, the processor uses the first defined value from the following list to set the device. The other values are ignored even if they are defined as well:

- The device in the context database that matches the `sc_device` parameter in the URL query string.
- The `device` attribute of the context site.
- The browser user agent associated with the current HTTP request.
- The `defaultDevice` attribute of the context site.
- The first device that has with the **Default** checkbox selected in the **Data** section from all devices that are assigned to the context item.



For example:

If you specify the `device` attribute in the website definition in the `Web.config` file, Sitecore uses it as a context device in all cases, except when the `sc_device` parameter is specified in the URL.

If you don't specify the `device` attribute, Sitecore uses the browser user agent to set the context device. If, for some reason, Sitecore cannot determine the user agent, it uses the `defaultDevice` attribute. If that attribute is not defined, Sitecore uses the first device that has with the **Default** checkbox selected in the **Data** section from all devices that are assigned to the context item.

LanguageResolver

This processor uses the first defined value from the following list to set the context language. The other values are ignored even if they are defined as well:

- The language that matches the `sc_lang` parameter in the URL query string.
- The language that is embedded in the URL, for example, `http://mysite.net/en/mypage`.

  This option only works if you set the `languageEmbedding` attribute of a link provider to a value other than `never`. The link providers are configured in the `Web.config` file, in the `linkManager` setting.

If none of these options are defined, Sitecore uses the first defined value from the following list to set the `Context.Language` property. Note that the logic which sets the `Context.Language` property is written in the property itself.

- The value of the `Sitecore Language` cookie associated with the context site.
- The value of the `language` attribute in the context site definition in the `Web.config` file.
- The value of the `DefaultLanguage` attribute of the `/configuration/sitecore/settings/setting` element in the `Web.config` file. For instance: `<setting name="DefaultLanguage" value="en" />`.

CustomHandlers

This processor triggers the custom handlers that are defined in the `<customHandlers />` section of the Web.config file.

QueryStringResolver

This processor works with the following parameters in the URL:

- `sc_itemid`

  If this parameter is specified, and if the following conditions are true, the processor sets the item with the specified ID as the context item:

  - The ID is valid.
  - The context item has not been already set.
  - Sitecore can find the item in the context database.
- `xmlcontrol`

  If this parameter is specified and contains a valid Sitecore shell control, Sitecore displays this control.

DynamicLinkResolver

If the context item has not already been set, this processor sets it by resolving the dynamic links that the `LinkManager` class generates.

AliasResolver

This processor sets the context item by resolving Sitecore aliases if the aliases are enabled.

The processor:

- Checks the requested URL to see if it matches an alias that exists in the system.

- Sets the Sitecore item that is associated with this alias as the context item if the context item is not defined yet.
- Processes the URL If the alias is associated with an external URL.

DefaultResolver

If the context item has not yet been set, this processor uses the “StartItem” and “RootPath” attributes of the context site to set the context item. The processor concatenates these values and gets the item that corresponds to the resulting path.

If the processor can’t use the `StartPath` argument to set the context item, it checks whether the access to the item that a user initially requested in the URL is denied by the Sitecore security system. If the latter is the case, the processor inserts an error message in the log file saying that the default item was not found.

FileResolver

This processor resolves the file system path for file items. If the file is resolved successfully, Sitecore returns it to the client.

ItemResolver

This processor resolves the Sitecore context item from the incoming URL if the item has not already been resolved. For example, it can use one of the following to resolve the item:

- The full absolute path (`http://your_site/sitecore/content/Home.aspx`).
- the display name.
- the path that is relative to the Home item.

If you want Sitecore to resolve items from your custom URLs, create a custom processor that inherits from `ItemResolver` and put it after ItemResolver in the pipeline.

LayoutResolver

This processor returns the presentation logic for the resolved item.

The cases when you may want to inherit from this processor to implement your custom logic include:

- Dynamic layout replacement logic. For example, you may want to use a field that a visitor fills in on the website to define a layout.

ExecuteRequest

This processor rewrites the context path and handles the “item not found” and “layout not found” errors.

The cases when you may want to inherit from this processor to implement your custom logic include:

- Custom redirects, for example, custom error pages.

Chapter 3

DMS Pipelines

The DMS introduces additional pipelines through the `Sitecore.Analytics.config` include file. It also adds processors to existing CMS pipelines.

This chapter contains the following sections:

3.1 Overview

This section describes the important pipelines that are specific to the Sitecore Digital Marketing System (DMS).

The Sitecore Digital Marketing System (DMS) uses the `/App_Config/Include/Sitecore.Analytics.config` file to define several pipelines and update pipelines in the `Web.config` file.

The DMS pipelines include:

- `automation` – Applies engagement automation.
- `startTracking` – Invokes the `initializeTracker` pipeline, parses tracking as specified by query string parameters, and processes the tracking defined for the context item.
- `initializeTracker` – Initializes the `Sitecore.Analytics.Tracker` class which is used to implement tracking.
- `parseReferrer` – uses the referrer HTTP header to identify search engines and search terms provided by the browser.
- `trafficTypes` – Identifies the traffic type associated with a visit — referral, organic search, branded search, and so on.
- `createVisit` – Records an analytics visit.
- `registerPageEvent` – Associates page events, including goals and failures, with pages.
- `triggerCampaign` – Triggers campaigns.
- `processMessage` – Processes outbound e-mail messages.

DMS updates the following CMS pipelines:

- `renderLayout` – Adds event handlers to support tracking.
- `getContentEditorWarnings` – Warns about items missing marketing profiles.
- `initialize` – Initializes tracking and engagement automation.
- `httpRequestBegin` – Initiates analytics diagnostics and sets the context items depending on page-level multivariate testing.
- `httpRequestEnd` – Applies analytics tracking.
- `sessionEnd` – Invokes rules and automations.

3.2 DMS Pipelines Involved in a Page Request

This section contains the important pipelines that are involved in a page request. The pipelines are listed in the order in which they are activated during a page request.

3.2.1 Initialize

This pipeline starts the background process which checks whether any of the automation states need to be processed at the current time.

It starts the automation worker process and GeoIP worker process.

Important processors:

`InitializeAutomation`

3.2.2 httpRequestBegin

The DMS uses the `Sitecore.Analytics.config` include file to add its processors in this `Web.config` pipeline.

The `StartDiagnostics` DMS processor starts the diagnostic process. The DMS adds its custom events for logging. These events copy all log messages to the Page Event table in the Analytics database.

Important processors:

`StartDiagnostics`

3.2.3 renderLayout

DMS adds its processors in this `Web.config` pipeline through the `Sitecore.Analytics.config` include file.

The `StartAnalytics` processor calls the `Tracker.StartTracking()` method which starts the actual tracking process. The method checks whether or not Analytics is activated and starts the `startTracking` pipeline.

Important processors:

`StartAnalytics`

3.2.4 startTracking

This pipeline creates a `Visitor` object and processes the URL query string and the context item.

Important processors:

- `TrackingFieldProcessor` – Processes an item and the goals and events that are associated with it.
- `ProcessQueryString` – Processes the `sc_camp` query string parameter.
- `InitializeTracker` – Starts the `InitializeTracker` pipeline.

3.2.5 InitializeTracker

This pipeline initializes the main Analytics objects, such as `Page`, `Visit` and `Visitor`.

- `Initialize` – Initializes the important classes such as Visit, Visitor, and Page. It checks whether or not this is a new visitor by checking if a cookie exists for the current visitor and visit. The cookies are:

  SC_ANALYTICS_GLOBAL_COOKIE

  SC_ANALYTICS_SESSION_COOKIE.

- `RunRules` – Runs the analytics rules defined in the following item: `/system/Settings/Analytics/Rules/Session Begin/Rules`.
- `RunAutomation` – Checks whether or not any of the automation states for the current visitor need to be processed with the current request.

3.2.6 loadVisitor

The processors in this pipeline load the data and the automation states that were recorded previously for the visitor.

Important processors:

- `LoadAnalytics` – If this visitor already exists, this processor loads the data that has already been collected for the visitor.
- `LoadAutomations` – Loads the automation states for the current visitor by executing the `AutomationManager.Provider.GetVisitorStates` method.

3.2.7 createVisit

If this is a new visit, the processors in this pipeline generate information about the new visit.

Important processors:

- `XForwardedFor` – Identifies the IP address of a client connecting to the web server.
- `UpdateGeoIpData` – Gets the location of a client connecting to the web server.
- `ParseReferrer` – Calls the `parseReferrer` pipeline to identify the referring URL.
- `SetTrafficType` – Calls the `trafficTypes` pipeline to determine the traffic type.

3.2.8 trafficTypes

This pipeline determines the traffic type.

3.2.9 insertRenderings

This pipeline contains the logic for the MV Testing and personalization.

3.2.10 httpRequestProcessed

The `EndAnalytics` processor in this pipeline:

- Stops the event logging process.
- Stops writing to the `Page Event` table in the `Analytics` database.
- Records how long the visitor was on the page.
- Records how long it took to process the request.

Important processors:

`EndAnalytics`

3.2.11 acceptChanges

Until this step, the changes are not written to the Analytics database, but are added to the `DataSet` class instead. The Processors in this pipeline check whether or not the changes that were made should be committed to the Analytics database.

3.2.12 commitDataSet

The `CommitAnalytics` processor in this pipeline writes the data to the `Analytics` database when the data is accepted in the `acceptChanges` pipeline.

The processor runs either at the `TrackerChanges.FlushInterval` interval, or when the row count exceeds the value `TrackerChanges.MaxRows.`

Important processors:

`CommitAnalytics`

3.2.13 registerPageEvent

If there are any page events defined for the context page, the processors in this pipeline run the appropriate page event rules or automation rules.

3.2.14 parseReferrer

This pipeline parses the keywords that users enter in search engines and uses them in personalization rules. To determine which search engines it should track, the pipeline uses the `engines` list:

```
…
<engines hint="raw:AddHostParameterName">
            <engine hostname="www.google" parametername="q"/>
…
```

Important processors:

`ParseGenericSearchEngine`