Dynamic Template System - Users Guide

How to use the Dynamic Template System

Christian Groth <kontakt@christian-groth.info>

Dynamic Template System - Users Guide: How to use the Dynamic Template System

by Christian Groth

Published 2009-01-12

Table of Contents

List of Tables

Chapter 1. Preface

This guide was written for all people who want to use the Dynamic Template System and need some help to get started. This first chapter deals with some general information about the DTS. Afterwards you'll hopefully get the DTS idea and understand the concepts, so you can deal with the sample project and finally get started with your own project.

What is it?

Dynamic Template System generates output based on templates. It takes a template and generates the content for all placeholders using parameters and generics. Afterwards the output can be just returned to any application using Dynamic Template System or saved into a file.

So why do you call it 'dynamic'? ... well the output format is not limited to xml, (x)html or whatever. You can generate output in any format you want. Perhaps you want to generate php, javascript or css files? Along with xhtml? All in one project? Just do it, it's really that easy as you will see in this guide.

Technologies

If you want to use the Dynamic Template System, you have to be able to handle some technologies. First of all you have to deal with xml. The configuration file for your project will be written in xml. The Dynamic Template System will read this file, build up a representation of your project in Java and process your commands. ... Commands? Where they come from? What is the java representation? Are you crazy? We'll cover all this in the next chapter, so far you just have to know that you have to be familiar with xml and the the format you want to create of course.

Let's just build up a list of technologies used by Dynamic Template System and keep in mind that you primarily have to deal with xml and your target technology:

- XML
- Target Technology
- Java

Chapter 2. Basic Concepts

There are some really basic concepts you have to know to be able to generate your content. This concepts won't get you really far and they you won't be able to do freaky things with them, but hey ... they're basics.

Dynamic Template System

You can look at the Dynamic Template System as your project. So if you have more than one project using this framework you will have more than one Dynamic Template System. A Dynamic Template System contains all of your projects information:

- Themes
- Pages
- Parameters
- Generics
- Processings
- States
- Insertion Patterns

All of these concepts will be handled here in this guide, but before we start with the basics just make one step back. What do we know so far?

- each project is a Dynamic Template System
- all information for one project are contained in theDynamic Template System
- all information are stored in one xml file

From XML to Java

To process all your Dynamic Template System information and generate the content a java representation of your xml must be created. After this is achieved the Dynamic Template System has some methods to generate the content. All these may be invoked using an Invoker. In general there are three possibilities to generate content:

- convert a single page
- export a single page
- export all pages

The last thing you have to know about conversion from xml to java is how boolean values (true, false or yes and no) are handled. If an optional boolean parameter is not there it will result in false, 'y', 'yes' or '1' will result in true and all other values will also result in false.

Batch Invoker

Dynamic Template System give you a class named BatchInvoker which is able to export all pages into the filesystem. This is the invoker also used in the sample project. You can start the invoker via command line and don't have to worry about anything but the parameters:

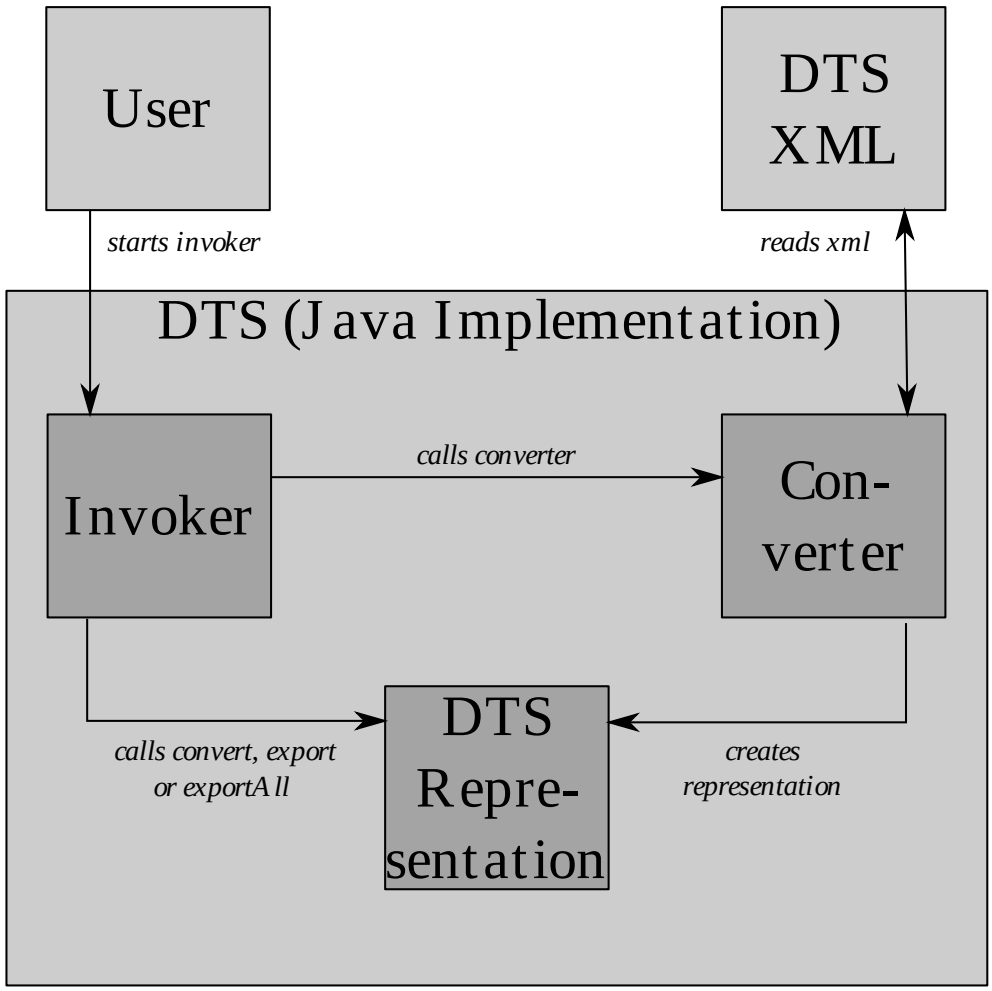
- basePath: the basePath for all relative pathinformation
- dtsFile: path to your Dynamic Template System xml file (relative to basePaht)

- exportPath: path to root export directory

When you start the BatchInvoker with this three information it will startup and (very roughly) do the following things:

- checks if base and export paths are existent and readable
- checks if the dtsFile is existent and readable
- initializes an converter
- export all pages using the converter>

The BatchInvoker also initializes the log4j system, so be sure to have a file called log4j.xml in the current directory (where you start the BatchInvoker from).



So now you should know how the Dynamic Template System roughly works and how you can export your pages. What you don't know is how to create the xml file and which possibilities you have.

Theme

A theme is basically a template. Each definition of a theme holds the following information:

- id
- template
- parameters

The id is used for pages to reference the theme they will base upon, but we'll discuss this in the next chapter. The template file is more interesting for now. This file can be of any non-binary format you like. It might be a plain text file, html, xml, sql or whatever. So here we have the first situation in which we are really dynamic.

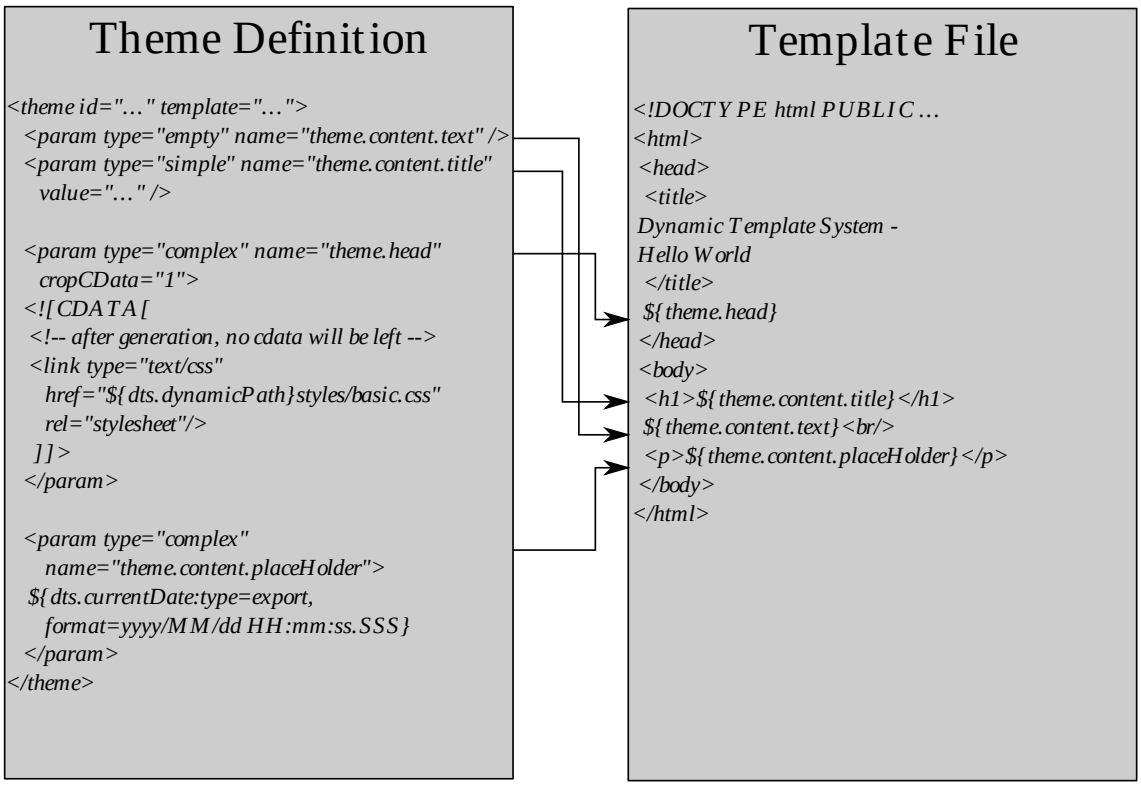
So what does a template do? It defines some sort of placeholders, right. So you nearly know how to create your first template-file to be used by a theme. Placeholders are just names with some special characters before and after the name, so that the Dynamic Template System might know where the placeholder really is.

To define a placeholder you just have to define a name with the prefix '${' and the postfix '}'. So if you want to define a placeholder called 'content' you have to write '${content}' in your template-file.

There is also an (optional) theme inheritance you might want to use. Instead of specifying a templateFile you can specify a superThemeId. You must not define both, so all themes contained in an inheritance (which only can have one root theme) uses the same file. This concept might be used to define different standard values for the same template file.

Table 2.1. general theme xml attributes

xml attribute	description	mandatory
id	unique id for theme	Y
template	path (relative to basePath) to template file	either template or superThemeId must be set!!
superThemeId	id of parent theme	either superThemeId or template must be set!!



Parameter

Parameters replace placeholders with concrete values. Each parameter has some basic attributes:

Table 2.2. general parameter xml attributes

xml attribute	description	mandatory
type	defines the parameter plugin type	Y
name	identifies the placeholder to be replaces	Y

The parameter name has to match some placeholder, so it defines the content position in the template file. The type specifies which concrete type of parameter is used. Dynamic Template System give you parameters for all types of concepts, these parameters are discussed later in this guide.

Page

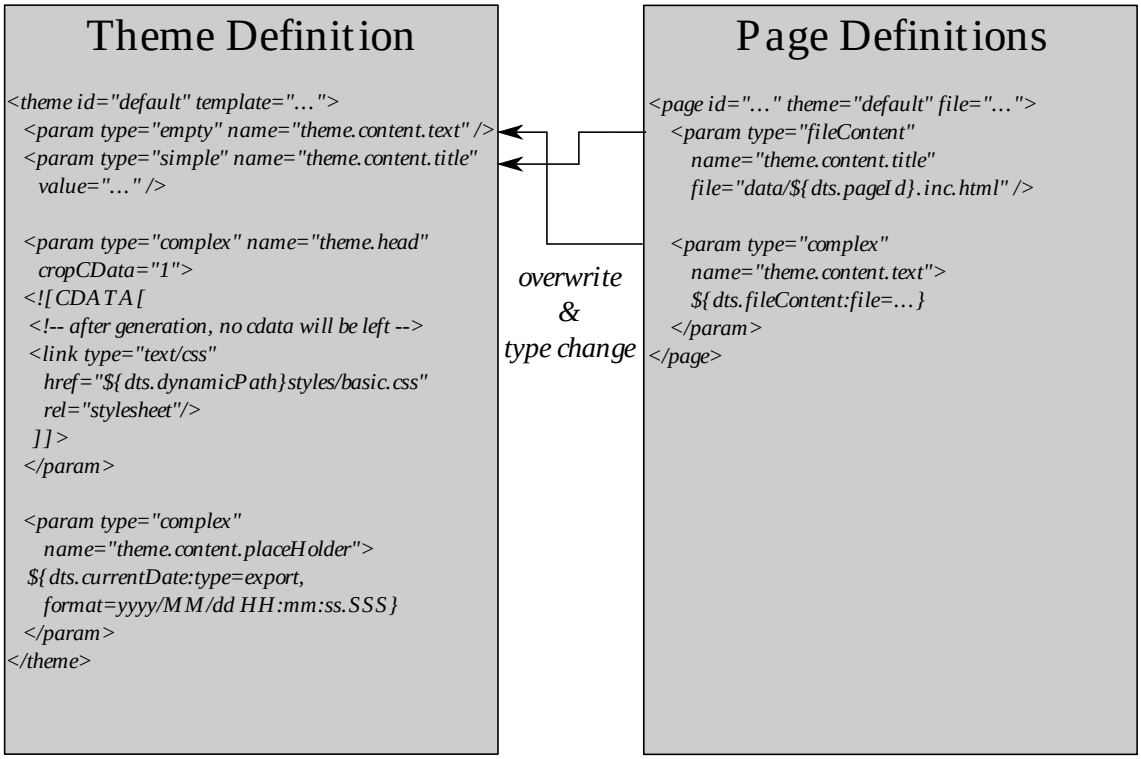
So far you know about themes and some basic parameters, but this is not enough to produce some real content. But don't worry, there's only one concept missing, the page.

A page will (mostly) result in a file if you call the exportAll (or export) method on any Converter/ Invoker. The basic information for every page are the following:

- id
- theme
- file
- parameters

The id and theme information are mandatory, so be sure to use a unique page identifier and set the theme that has to be used for this page. Of course a theme with the specified id must be defined.

Every parameter you define must override a parameter which is existent in the referenced theme. So it's always a good idea to define every placeholder as parameter in the theme and override them in a page. Therefore the empty parameter is very useful to be used in the theme definition. If you try to define a non-overriding parameter in a page this will result in an exception, because the theme does not offer this parameter.



The last 'rule' you have to keep in mind is: The latest definition wins. So if you define parameter 'a' with the value '1' in the theme definition and also parameter 'a' with value '2' in the page definition the generated value will be '2'.

There is one point we have to anticipate. You might also leave away the file attribute in your page xml definition. This will result in an not exportable page. So be sure that you wont use the export functionality of any Invoker with such a page-id. If you're using exportAll this kind of pages will be ignored.

So why do you want to create such kind of pages?? Well ... as you might see later there is a page-include concept which lets you use any page, convert it on the fly and include it as parameter or generic value. But we'll get this later.

Table 2.3. general page xml attributes

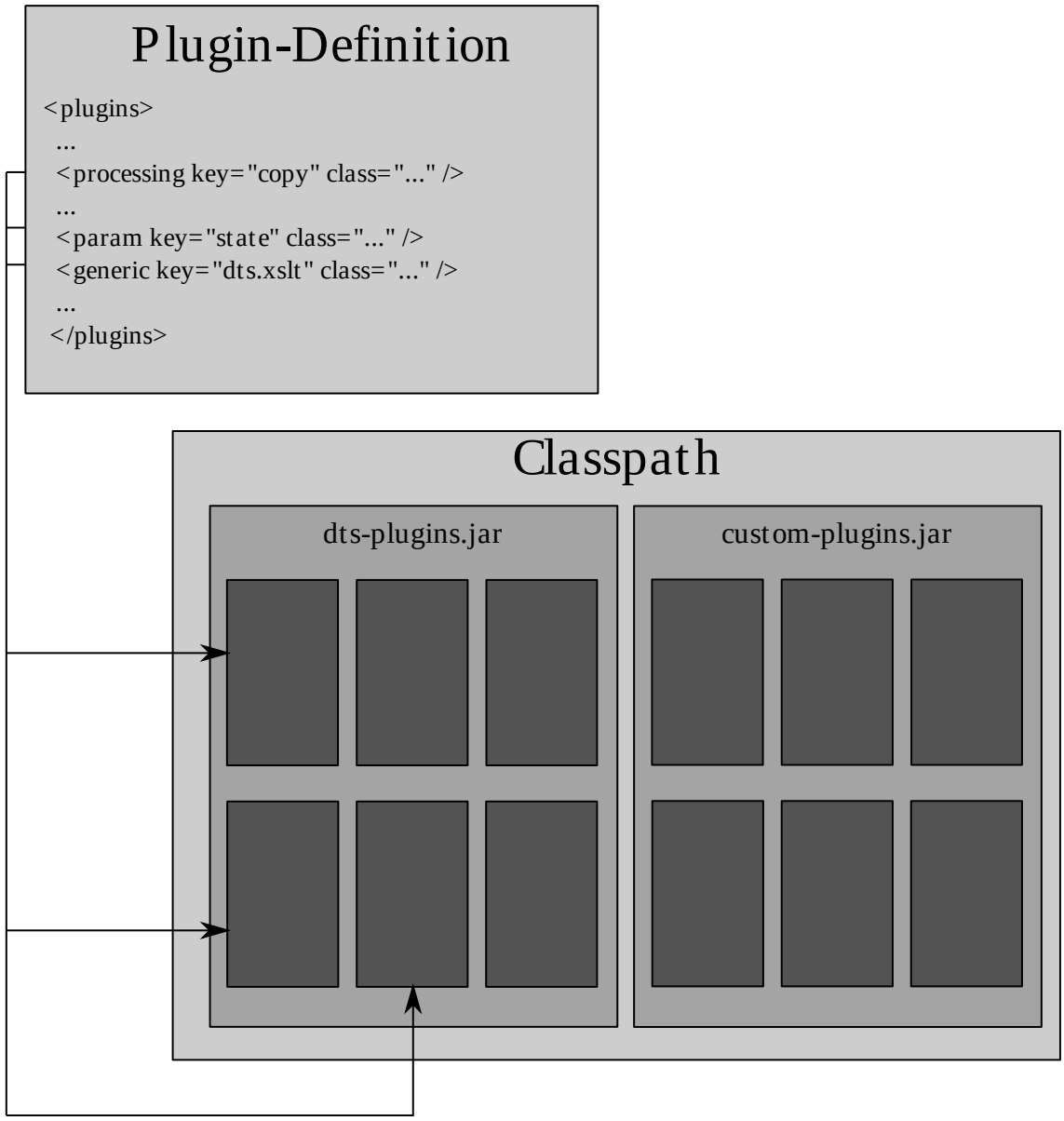
xml attribute	description	mandatory
id	unique id for page	Y
theme	id of theme to be used	Y
file	path (relative to exportPath) to output file	N

Chapter 3. Advanced Concepts

So far you have all information you need to generate content with basic concepts. But there is more of course. All the additional stuff you can use to generate content and be more dynamic will be discussed in this chapter.

Plugins

The Dynamic Template System works with a plugin system, so you can add any self developed plugins or even leave unneeded plugins away. if you take a look in the dts xml file you'll find a block called 'plugins'. Each type of plugin has a 'key' and a 'class' attribute. For all standard plugins an entry will be provided within the sample project, but of course you're free to change the key if you want to. On the one hand the key attribute of every plugin type is really important for your project, on the other hand the class attribute is only used internally.



If you use e. g. a parameter you know that you have to specify a type for that parameter. The value for the type attribute is the value of the key attribute of any parameter plugin. So if you register the

plugin class for the simple type parameter with the key 'complex', then every parameter you use with the type 'complex' will be a parameter of type simple. So as you can see it's always a good idea to use a key that fits the class.

Of course the same rules apply to all other plugins, as you'll see in the following chapters.

Table 3.1. general plugin xml attributes

xml attribute	description	mandatory
id	identifier for this plugin to be used as parameter, generic or processing	Y
class	full qualified classname for internal use only (do not change, unless you really know how the stuff works)	Y

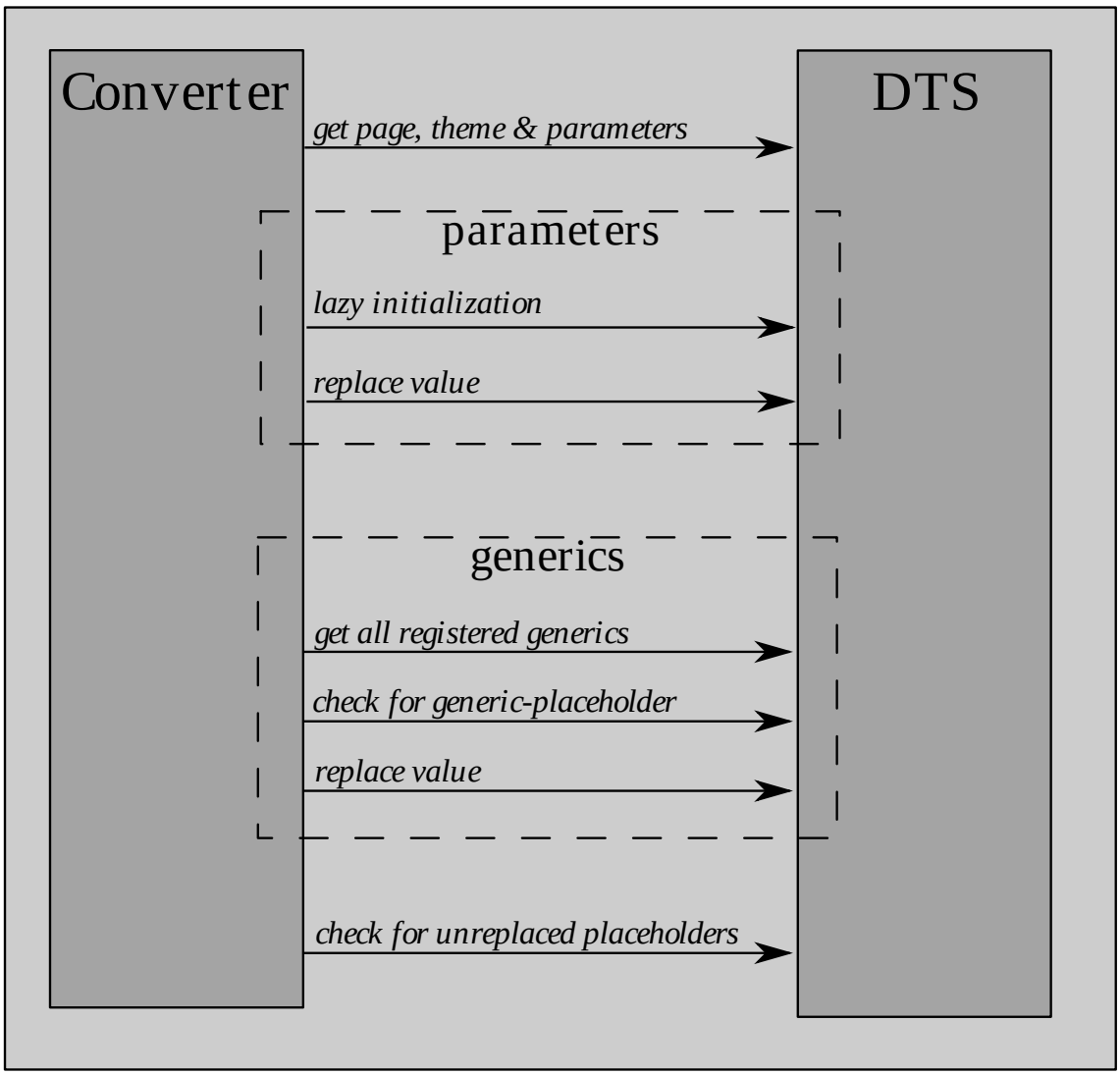
Generics

Generics are very similar to parameters and placeholders in template files, but you can define a generic everywhere you want. A first let's take a look how generics are replaced by concrete values. For now think of generics just like placeholders which will be replaced by something.

Converting A Page

If the Dynamic Template System converts a page there are a few steps which will be done.

1. collecting all parameters (themes and page)
2. processing parameters
3. processing generics
4. checking for unreplaced parameters/generics



As you can see generics are processed after parameters. Also there is no real definition for any generic. A generic is written exactly like a placeholder in a template file, but the name of the placeholder must match a key of a generic plugin.

There is only one difference between a placeholder and a generic. The generic can have some parameters or arguments. In general generics are designed to create output as an independent component, e. g. date/time values. Generic parameters are used to change the mode or output of the generic, in fact generics are not page or theme dependent like parameters are.

If you want to or must provide parameters to a generic you have to do just a few things:

- add ':' after the generic name (the plugin key) to start the parameter section
- add your parameters as 'name'='value' pair
- separate parameter name/value pairs with ','

So if you want to pass teh parameter 'foo' with the value 'bar' and another one called 'foo2' with the value 'bar2' to a generic called 'myGeneric' use the following syntax: '${myGeneric:foo=bar,foo2=bar2}'

Of course Dynamic Template System delivers some basic generics like it also does with parameters. You'll find some generics which also can be used as parameter like file content, xsl and so on. All build-in generics are discussed in an later chapter.

Lazy Initialization

Another big advantage of generics is that parameters (better, one or more of their attributes) may be lazy initialized ... lazy what?

If you think of a parameter as discussed above you know that you have to specify a type, a name and some additional attributes depending on the specified type. So if you want to use the parameter of type simple, you have to specify the 'value' attribute. (as you will see below) Normally you would specify the vlue as text in the xml-attribute. But what if you want that value to be ... well generic? Different value for different pages ...

Well .. sounds you would like to use a state instead (see some lines below). So you define a state and different conditions, but you have to specify a condition for each page. So the condition value of the state parameter is some kind of generic. Well ... generic ... sounds like generics ... might we use them here? Might we even put a generic in the xml-attribute?? ... Yes!

So lazy initialization means parameter attributes which contain a generic which must be replaced before the parameter value can be computed. But take care, not every parameter type is capable of lazy initialization and even if it is, not every of its attributes is it. Below in the build-in parameters section you'll find a column which says if a parameter attribute is capable of lazy initialization.

Non Deterministic Parameters and Generics

Parameters and Generics may be marked as non deterministic. Such Parameters and Generics will produce different results when set up with the same arguments but executed at different times. This information is planned to be used for some sort of caching. With this cache the Dynamic Template System will be able to only convert and/or export pages which have changed, but hey ... it's scheduled for the future.

States

What is a state? ... You can think of a state as being one big 'if-then-else'. You specify different outputs bound to different conditions. So if you put in 'conditionA' the output might be '1', if you put in 'condition2' the output might be 'B' or whatever.

So in fact a state is something like a blackbox with an condition input that generates some reproducible output. Basically you have to define the following information:

- id
- cropCData
- pre value
- post value
- default value
- conditions

As you might already guessed, the id is used to identify the state and therefore has to be unique. The optional cropCData flag indicates if cdata sections might be cropped for pre/post/default-value and the condition values.

The pre/post/default-values are specified as child node named 'pre', 'post' and 'default'. they are all three optional. So if you do not define a default value and pass an unknown condition the application will fail to resolve the value. That's not good, so if you do not define a default value, be sure to pass only known conditions.

You have to specify at least one condition. A Condition contains (as xml content) the output value and holds its condition-value as xml attribute 'condition'.

Table 3.2. general state and contidion xml attributes

xml attribute	description	mandatory
id	unique identifier for this state	Y
cropCData	flag if cdata section syntax will be cropped	Y
value (attribute of condition node)	condition value	Y

InsertionPatterns

An insertion pattern is very similar to the content of a template file and is used quite similar to the pageInclude parameter/generic (as you might see later). You might specify the following information:

- id
- cropCData
- default values for active/inactive/unreplaced insertion points
- the data

Well ... 'id' and 'cropCData' might be well known by now, so lets focus on the big data part. The xml content of the 'data' child node holds the patterns data. This data contains placeholders, similar to placeholders in a template file. They differ only in their prefix, because for an insertion pattern you have to use the prefix '#{' instead of '${'. This is necessary because name clashes with generics should be avoided.

Each of this placeholders might be set to active or inactive ... or it might even not be set. So you wont replace a placeholder with concrete output data. In fact you have three different modes per placeholder, or insertion point.

As you might see later, you can specify the values for active, inactive and unreplaced insertion points using parameter or generic. Furthermore you can specify a default value for each of this situations as child node with the names 'defaultActiveInsertion', 'defaultInactiveInsertion', 'defaultUnreplacedInsertion'.

Processings

Processings are only relevant if the 'exportAll' method is used. This method runs all registered processings in two phases. In general the exportAll mode runs in three phases:

1. run pre-processings
2. convert and export all pages
3. run post-processings

Processings are registered in the xml file as child of the 'dynamicTemplateSystem' node under 'processings/pre' and 'processings/post' and are processed in their from top to bottom.

In general processings are not meant to do any of the 'core work' but prepare (pre-processings) and/or clean up (post-processings). Because of this, there is a convention that processings will only read from basePath (and subdirectories) but never write to it. The exportPath with all its subdirectories are allowed to be read and written to. Later in this guide you will get to know all build-in processings and if you take a look in the sample project you can see them in action.

Chapter 4. Build-In Parameters

This chapter lists and briefly explains all build-in parameters that can be used out of the box. At the end of this section you'll find a table with all parameter classnames which have to be set up in the plugins section in your dts xml. The 'lazy?' column says if this parameter attribute capable of lazy initialization.

Parameter - Empty

An empty parameter whether defines any additional attributes, nor any value. It will result in nothing as its value, so any placeholder will be replaced by nothing. You can use this type of parameter as kind of disabled parameter/placeholder in combination with parameter overriding which is discussed later in the page section.

Table 4.1. empty parameter xml attributes

xml attribute	description	mandatory	lazy?
-	-	-	-

Parameter - Simple

The simple parameter defines an additional xml attribute called 'value', which holds ... the value, yes. So this one is really a simple parameter.

Table 4.2. simple parameter xml attributes

xml attribute	description	mandatory	lazy?
value	the value to be used	Y	N

Parameter - Complex

Every parameter of type complex reads its value from the xml content of the parameter node. So you have the possibility to generate structured content (xhtml, xml). It also defined the optional attribute 'cropCData' which holds a boolean value and tells if potential cdata sections of the node-content may be replaced.

Table 4.3. complex parameter xml attributes

xml attribute	description	mandatory	lazy?
cropCData	flag if cdata section syntax will be cropped	Y	N

Parameter - File Content

You might want to use the content of another file as parameter value? No problem, this type of parameter is exactly what your looking for. If defines just one xml attribute named 'file' which holds the relative path to the file being included.

Table 4.4. file content parameter xml attributes

xml attribute	description	mandatory	lazy?
file	path (relative to base path) to the file to be included as content	Y	Y

Parameter - Page Include

Takes a page by id in 'page' attribute, converts the referenced page and includes the result.

Table 4.5. page include parameter xml attributes

attribute	description	mandatory	lazy?
page	id of page to be included	Y	Y

Parameter - Properties

This type of parameter reads a property file referenced by the 'properties' attribute and resolved the value to the specified property key in the 'key' attribute.

Table 4.6. properties parameter xml attributes

xml attribute	description	mandatory	lazy?
properties	path (relative to base path) to the properties file	Y	Y
key	property key, the value will be used as output	Y	Y

Parameter - XSLT

You can use a xsl file to transform a xml document and use the result as parameter value. Specify the path to the xsl file with the 'xsl' attribute and the path to the xml document via 'xml' attribute.

Table 4.7. xslt parameter xml attributes

xml attribute	description	mandatory	lazy?
xml	path (relative to base path) to xml document	Y	Y
xsl	path (relative to base path) to xsl document	Y	Y

Parameter - State

This parameter uses a state to provide it's value, so you have to specify the state and the condition to be used. The output value will be the result of the states condition.

Table 4.8. state parameter xml attributes

xml attribute	description	mandatory	lazy?
state	id of state to be used	Y	Y
value	states condition	Y	Y

Parameter - Insertion Pattern

Uses an insertion pattern and sets active and inactive insertion points.

Table 4.9. insertion pattern parameter xml attributes

xml attribute	description	mandatory	lazy?
insertion pattern	id of insertion pattern to be used	Y	Y
active	comma separated list of active insertion points	N	Y

xml attribute	description	mandatory	lazy?
inactive	comma separated list of inactive insertion points	N	Y
activeValue	value to be set for active insertion points (will override default value, if set)	N	Y
inactiveValue	value to be set for inactive insertion points (will override default value, if set)	N	Y
unreplacedValue	value to be set for unreplaced insertion points (will override default value, if set)	N	Y

Table 4.10. parameter plugin classnames

parameter plugin	classname
empty	de.groth.dts.plugins.parameters.ParameterTypeEmpty
simple	de.groth.dts.plugins.parameters.ParameterTypeSimple
complex	de.groth.dts.plugins.parameters.ParameterTypeComplex
file content	de.groth.dts.plugins.parameters.ParameterTypeFileContent
page include	de.groth.dts.plugins.parameters.ParameterTypePageInclude
properties	de.groth.dts.plugins.parameters.ParameterTypePropertyReader
xslt	de.groth.dts.plugins.parameters.ParameterTypeXslt
state	de.groth.dts.plugins.parameters.ParameterTypeState
insertion pattern	de.groth.dts.plugins.parameters.ParameterTypeInsertionPattern

Chapter 5. Build-In Generics

This chapter lists and briefly explains all build-in generics that can be used out of the box. At the end of this section you'll find a table with all parameter classnames which have to be set up in the plugins section in your dts xml.

Generic - Current Date (non deterministic)

This generic processes date and time information using different modes, processing

- current: the current date/time during processing
- convert: the date/time the currently processed page was converted last time
- export: the date/time the currently processed page was exported last time
- exportAll: the date/time the exportAll method was used

Additionally you have to specify the format parameter. This one describe sthe output format using java.text.DateFormat symbols, which can be found in latest javadocs.

Table 5.1. date generic attributes

attribute	description	mandatory
type	mode to be used (see list above)	N (current used as default)
format	date/time output format	Y

Generic - Dynamic Path

A generic to provide the relative path to the root-directory. So if the currently converted page has a file attribute like "dirA/dirB/file" this generic will result in "../../".

There is an optional parameter named 'pageId' which takes a page id and appends the path from the given page, so that you might link all pages without wondering about directory depths.

Table 5.2. dynamic path generic attributes

attribute	description	mandatory
pageId	path to output file of referenced page will be appended to output	N

Generic - File Content

Includes the content of the specified file. The parameter 'file' is evaluated as relative path from baseDtsPath.

Table 5.3. file content generic attributes

attribute	description	mandatory
file	path (relative to basePath) to be included	Y

Generic - Page ID

Prints out the id of the page currently converted.

Table 5.4. pageId generic attributes

attribute	description	mandatory
-	-	-

Generic - Page Include

Takes a page by id in 'page' parameter, converts the referenced page and includes the result.

Table 5.5. page include generic attributes

attribute	description	mandatory
page	id of page to be included	Y

Generic - Properties

Reads the specified properties file and prints out the value to the specified key.

Table 5.6. properties generic attributes

attribute	description	mandatory
properties	path (relative to basePath) to properties file	Y
key	key of properties file	Y

Generic - XSLT

Processes a xml transformation using the specified xml and xsl documents.

Table 5.7. xslt generic attributes

attribute	description	mandatory
xml	path (relative to basePath) to xml document	Y
xsl	path (relative to basePath) to xsl document	Y

Generic - State

Uses a state to provide its value.

Table 5.8. state generic attributes

attribute	description	mandatory
state	id of state to be used	Y
value	states condition	Y

Generic - Insertion Pattern

Transforms an insertion pattern and outputs the transformed data as value. All parameters which not match one of the table below (except the last entry of course) will activate or deactivate an insertion point named like the parameter. Use a boolean representation of 'true' to activate, all other parameter values will result in an deactivated insertion point. Only not mentioned insertion points will get the state 'unreplaced'.

Table 5.9. insertion pattern generic attributes

attribute	description	mandatory
insertion pattern	id of insertion pattern to be used	Y
activeValue	value to be set for active insertion points (will override default value, if set)	N
inactiveValue	value to be set for inactive insertion points (will override default value, if set)	N
unreplacedValue	value to be set for unreplaced insertion points (will override default value, if set)	N
any parameter which is not one from the above list	put a boolean representation to active or deactive the insertion point. 'pointA=1,pointB=0' will result in insertion point 'pointA' active and 'pointB' inactive.	N

Table 5.10. generic plugin classnames

generic plugin	classname
current date	de.groth.dts.plugins.generics.GenericTypeCurrentDate
dynamic path	de.groth.dts.plugins.generics.GenericTypeDynamicPath
file content	de.groth.dts.plugins.generics.GenericTypeFileContent
page id	de.groth.dts.plugins.generics.GenericTypePageId
page include	de.groth.dts.plugins.generics.GenericTypePageInclude
properties	de.groth.dts.plugins.generics.GenericTypePropertyReader
xslt	de.groth.dts.plugins.generics.GenericTypeXslt
state	de.groth.dts.plugins.generics.GenericTypeState
insertion pattern	de.groth.dts.plugins.generics.GenericTypeInsertionPattern

Chapter 6. Build-In Processings

This chapter lists and briefly explains all build-in processings that can be used out of the box. At the end of this section you'll find a table with all parameter classnames which have to be set up in the plugins section in your dts xml.

Processing - Clear

Clear a directory, meaning all subdirectories and files will be deleted, but not the directory itself. The given path will be evaluated relative to the exportPath.

Table 6.1. clear processing xml attributes

xml attribute	description	mandatory
path	path relative to exportPath	Y

Processing - Copy

Will copy resources (directory or file) from basePat into exportPath and create target subdirectories if not existent.

Table 6.2. copy processing xml attributes

xml attribute	description	mandatory
from	resource source path relative to basePath	Y
to	resource target path relative to exportPath (directories will be created if not existent)	Y

Processing - Delete

Will delete resources (directory or file). Given path will be evaluated relative to exportPath.

Table 6.3. delete processing xml attributes

xml attribute	description	mandatory
path	path relative to exportPath	Y

Processing - Generate Config

Will generate the dts xml file (which is your projects 'configuration'). The given path will be evaluated relative to exportPath.

Table 6.4. generate config processing xml attributes

xml attribute	description	mandatory
path	path relative to exportPath	Y
file	filename to be used for generated xml file	Y

Table 6.5. processing plugin classnames

processing plugin	classname
clear	de.groth.dts.plugins.processings.ProcessingTypeClearDirectory

processing plugin	classname
copy	de.groth.dts.plugins.processings.ProcessingTypeCopyResources
delete	de.groth.dts.plugins.processings.ProcessingTypeDeleteResources
generate config	de.groth.dts.plugins.processings.ProcessingTypeGenerateConfig

Chapter 7. Sample Project

At this point of this guide you have an idea of all the concepts used by Dynamic Template System and all build in parameters, generics and processings. You might have asked yourself how to get all the stuff working and producing some output, right?

Well .. take a look at the sample project. This Dynamic Template System project uses all concepts and all build in plugins, so you can see them in action and also get an idea how they are meant to be used.

Structure

Let's focus on the directory structure first. If you unzip the archive (let's call this directory 'sampleProject') you get the following directories and files:

Table 7.1. sample project directory structure

resource	description
sampleProject/lib	all needed libraries plus all dts modules are put here
sampleProject/sample	this is the project directory, later more to this one ...
sampleProject/src	you can put your won plugin here, they will be put to classpath before execution. Nice service eh?
sampleProject/export	directory with generated content (will be created on first run)
sampleProject/build.VERSION	a file that shows you the current version of used Dynamic Template System
sampleProject/configureClasspath.bat	hanges classpath (temporarily) before execution
sampleProject/ executeWithNoCustomSources.bat	execute the BatchInvoker and exports all pages to export-folder. Note that the BatchInvoker is executed directly. If you want to use your own plugins from sampleProject/src use execute.bat instead (see one entry below).
sampleProject/execute.bat	execute the BatchInvoker and exports all pages to export-folder. Note that before the BatchInvoker is executed the ant script will be run, to compile your sources to sampleProject/ lib (see one entry below)
sampleProject/build.xml	compiles your sources from sampleProject/src and puts them into sampleProject/lib
sampleProject/log4j.xml	log4j settings

As you already know, you have to provide three parameters to the BatchInvoker so set basePath, dts xml file and exportPath. If you want to change these parameters, you can find them in the execute.bat and executeWithNoCustomSources.bat.

The project-directory (sampleProject/sample, further called workspace) is organized within the structure you see below. Of course you can organize this directory like you want, but it is a good practice to provide the projects dts xml in the root of your workspace. It's also a good idea to separate data for content generation, templates and static resources.

Table 7.2. sample project workspace directory structure

resource	description
workspace/data	data used for generating content
workspace/static	static resources (these are copied into export directory during processing)

resource	description
workspace/themes	template files for all themes can be found here
workspace/helloWorld.dts.xml	your projects dts xml file

Before you take a closer look at the helloWorld project definition, let me say one thing: It's possible to do things easier.

Yes, some things in this helloWorld project might be done a little complicated, but this example uses each element (parameter, generic, processing) at least one time, so that you can take a look at how the different elements are meant to be used.

If you run one of the batch files of call the BatchInvoker from console or do whatever to start this Dynamic Template System project you'll notice a new directory 'export' (mentioned above) and log4j logfiles. For now just ignore the logfiles and take a look into the export folder.

You will see one html file per defined page in the project definition. Each html file used only one concept or parameter or generic, so that you have the possibility to study the xml definition, maybe think about it and think about what you expect to happen and compare it with the html result.

Chapter 8. Get Started

To get started with your own Dynamic Template System project there is no general way which fits all requirements. This chapter can not take your hand and tell you what steps have to be done to get to you perfect project setup. I just want to list some notes and hints what you might think of if you set up your own Dynamic Template System project.

- it's always a good idea to take a look at a sample or helloWorld project at first
- you might want to copy the helloWorld.dts.xml, rename it or just copy the plugins definition in your own dts xml file
- it's helpful to keep your workspace clean. put the dts xml in the root directory and create different folders for different information (static files, template files, include files, ...)
- copy the log4j.properties to get a log file so you can take a look at it, if something fails during execution