Flat File Adapter Reference

This page explains the **Flat File Adapter** in Bridge context. If you were looking for the same information regarding the PAS Designer, refer to Flat File Adapter in the Designer guide.

Tagged Values

Class <<FlatFile>>

This is the root record of the flat file declaration. This class cannot have any attributes, only associations to <<FlatFileRecord>> classes and <<FlatFileGroup>> classes are allowed.

Tagged Value	Description	Allowed Values		Example
escapeCharacter	Defines the character used for escaping when a reserved character is used within a field value.	any character		`/`
fillCharacter	Defines a dummy character to fill non-existent values (results in NULL). Used for fixed attribute layout only.	any character		`0`
quoteCharacter	The **quoteCharacter** will be ignored by reading field value.	any character		`/`
recordSeparator	Separator of the different records, normally line feed and carriage return. For serialized files, any other character can be defined.	any character or one of		`<newline>`
		`<esc>`		
		`<newline>`	parses correctly on Unix and Windows platforms	
		`<space>`		
		`<tab>` `<tabulator>`		
		`<unixnewline>`	composes newline	
		`<windowsnewline>`	composes newline and carriage return	
reservedCharacters	Defines a list of characters to be escaped automatically when the file is composed.	any character		`{"/", "%", "&", "(", ")"}`

Class <<FlatFileRecord>>

This class describes the attributes of a record. A record is one "line" in a file and can consist of multiple attributes.

Tagged Value	Description	Allowed Values		Example
attributeLayout	Defines the attribute layout (**fixed** or **separated**) of the flat file record. • **Fixed**: For attribute values that are shorter than the maximum length of the field, the field is filled with a fill character. As per	`fixed`	fixed attribute layout	

	default, this is blank space, but you can change the fill character in tagged value **fillCharacter** of the <<FlatFile>> class (see Tagged Values of Class <<FlatFile>>). • **Separated**: If separated is used, specify the separator using **attribute Separator** (see below).	separated	separated attribute layout	
attributeSeparator	Defines the attribute/field separator.	any character	use this character as attribute separator	
		<Tab>	use tabulator as attribute separator	
attributePattern	A RegEx pattern to parse the record content into the attributes using capture groups.	a valid regular expression		`^(.?)(://)([A-Za-z0-9.])(:[0-9])(/.)$`
evaluationOrder	Defines the order in which the association of the classes starting on same parent class must be processed, see Flat File Adapter Parsing Process.	any integer		
ignoreEmptyRecords	Boolean value for ignoring empty records. If set to true, no item will be generated, if none of the defined attributes or sub records have any content. Note, that a record containing only empty **Strings** is not empty – in opposition to a record composed from `NULL`s. See **ignoreEmptyStrings** below to skip processing of records containing only empty **Strings**.	true	ignore empty records	
		false (default)	process empty records	
ignoreEmptyStrings	Boolean value for ignoring empty string attributes. If set to **true**, empty string values will be processed to `NULL`. Use this tag in combination with **ignoreEmptyRecords** to skip processing of records containing only empty **Strings**.	true	ignore empty string values	
		false (default)	preserve empty string values	
lineNumber	Specifies the number of a record in the file. The first record is lineNumber=1, the second lineNumber=2, etc.	any integer		
pattern	A pattern to identify the record. The pattern is checked before the fields are separated. If no pattern is defined, all records will be parsed.	any character		
		a valid regular expression		`^Pattern.*`

suppressEscaping	Boolean value to suppress escaping. If **suppressEscaping** on a <<FlatFileRecord>> is true, <<FlatFileComplexAttribute>> that are part of this record will inherit this setting.	true	attribute values of this record will not be un-escaped (parser) or escaped (composer)	
		false	escaping /un-escaping is not suppressed	
parseMacro **composeMacro**	A macro that is executed while parsing/composing a file or complex field. This macro can contain multiple commands separated by commas or spaces. Macros on classes are executed before the processing of its attributes or associations. The ID represents a counter. The following counters are available: • eight **automatic counters** with ID `AUTO0` .. `AUTO7` • two **automatic line counters** with ID `LINE0` and `LINE1` (parsing only) • unlimited **custom counters** with ID `CUSTOM0` .. `CUSTOMx` Automatic counters are increased by 1 for each processed record. Custom counters have to be increased manually using the increase macro. All counters have the initial value of 0 when they process the first record. For more details on macro commands see Macro Expressions .	any valid macro expression (see Macro Expressions)		`GetCounter(AUTO0)`

Class <<FlatFileGroup>>

This class is used to group multiple records into one (virtual) structure. A group does not have a representation in the flat file itself.

Tagged Value	Description	Allowed Values	Example
pattern	A pattern to identify the record. The pattern is checked before the fields are separated. If no pattern is defined, all records will be parsed.	any character	`^Pattern.*`
		a valid regular expression	

Class <<FlatFileSubRecord>>

Tagged Value	Description	Allowed Values	Example
condition	A condition that must evaluate *true* if the record exists. The condition can refer to a self object which represents the current state of the parent.	any valid conditional expression	`self.UNS.exists()`

evaluation Order	Defines the order in which the associations starting on same parent class must be processed, see Flat File Adapter Parsing Process.	any integer	
offset	Define the position of this record in the flat file, starting with 0 for the first record and always relative to the parent element.	any integer	

Attribute <<FlatFileAttribute>>

Fixed Layout

Tagged Value	Description	Allowed Values		Example
decimals	Replaced by **format.**			
externalLength	Number of characters of the field (only for fixed length records relevant).	any integer		
format	Pattern for formatting numeric and date & time values. For details see Number Formatting respectively Date and Time Formatting.	any valid number or dateTime pattern		`S9G999G990D00` `%Y.%m.%d-%H:%M:%`
nativeTyp deprecated	Replaced by **format**.			
order	The evaluation order of the attributes. If **offset** is not used, order reflects the field number within the record.	any integer		
offset	The character position of this field within the record.	any integer		
padding	Defines the padding rule for the field. When parsing, the characters on the left or right side are ignored up to the first different character. When composing, the field is filled on the left or right side with the specified character.	left("<any character>")		`left("0")`
		right("<any character>")		`right(" ")`
suppressEscaping	Boolean value to suppress escaping.	true	attribute values of this attribute will not be un-escaped (parser) or escaped (composer)	
		false	escaping/un-escaping is not suppressed	
parseMacro **composeMacro**	A macro that is executed while parsing/composing a file or complex field. This macro can contain multiple commands separated by commas or spaces. Macros on classes are executed before the processing of its attributes or associations. The ID represents a counter. The following counters are available: • eight **automatic counters** with ID `AUTO0` .. `AUTO7` • two **automatic line counters** with ID `LINE0` and `LINE1` (parsing only) • unlimited **custom counters** with ID `CUSTOM0` .. `CUSTOMx` Automatic counters are increased by 1 for each processed record. Custom counters have to be increased manually using the increase macro. All counters have the initial value of 0 when they process the first record. For more details on macro commands see Macro Expressions .	any valid macro expression (see Macro Expressions)		`GetCounter(0)`

Separated Layout

Tagged Value	Description	Allowed Values	Example

format	Pattern for formatting numeric and date & time values. For details see Number Formatting respectively Date and Time Formatting.		
order	The evaluation order of the attributes. If **offset** is not used, order reflects the field number within the record.	any integer	
offset	The relative position of the field in respect of the other fields in the record, e.g. field number 3 has **offset** = 2.	any integer	
suppressEscaping	Boolean value to suppress escaping.		
parseMacro **composeMacro**	A macro that is executed while parsing/composing a file or complex field. This macro can contain multiple commands separated by commas or spaces. Macros on classes are executed before the processing of its attributes or associations. The ID represents a counter. The following counters are available: • eight **automatic counters** with ID `AUTO0` .. `AUTO7` • two **automatic line counters** with ID `LINE0` and `LINE1` (parsing only) • unlimited **custom counters** with ID `CUSTOM0` .. `CUSTOMx` Automatic counters are increased by 1 for each processed record. Custom counters have to be increased manually using the increase macro. All counters have the initial value of 0 when they process the first record. For more details on macro commands see Macro Expressions .	any valid macro expression (see Macro Expressions)	`GetCounter(0)`

Attribute <<FlatFileComplexAttribute>>

This class is used to divide fields into sub-fields. You can think of this like a <<FlatFileRecord>> placed within a single field.

Tagged Value	Description	Allowed Values		Example
attributeLayout	Defines the attribute layout (**fixed** or **separated**) of the complex attribute. • **Fixed**: For attribute values that are shorter than the maximum length of the field, the field is filled with a fill character. As per default, this is blank space, but you can change the the fill character in tagged value **fillCharacter** of the <<FlatFile>> class (see Tagged Values of Class <<FlatFile>>). • **Separated**: If separated is used, specify the separator using **attributeSeparator** (see below).	fixed	fixed attribute layout	
		separated	separated attribute layout	
attributeSeparator	Defines the attribute/field separator.	any character	use this character as attribute separator	
		<Tab>	use tabulator as attribute separator	

attributePattern	A RegEx pattern to parse the field content into a complex structure using capture groups.	a valid regular expression	`^(.?)(://)([A-Za-z0-9.])(:[0-9])(/.)$`	
suppressEscaping	Boolean value to suppress escaping.	true	attribute values of this attribute will not be un-escaped (parser) or escaped (composer)	
		false	escaping/un-escaping is not suppressed	
parseMacro **composeMacro**	A macro that is executed while parsing/composing a file or complex field. This macro can contain multiple commands separated by commas or spaces. Macros on classes are executed before the processing of its attributes or associations. The ID represents a counter. The following counters are available: • eight **automatic counters** with ID `AUTO0` .. `AUTO7` • two **automatic line counters** with ID `LINE0` and `LINE1` (parsing only) • unlimited **custom counters** with ID `CUSTOM0` .. `CUSTOMx` Automatic counters are increased by 1 for each processed record. Custom counters have to be increased manually using the increase macro. All counters have the initial value of 0 when they process the first record. For more details on macro commands see Macro Expressions .	any valid macro expression (see Macro Expressions)		`GetCounter(0)`

Action "parse"

Tagged Value	Description	Allowed Values

resource	Instead of specifying parameters **name** or **data** for the parse flat file action (see section Parameters below), you can import the file to be parsed as a resource to your Builder project (see Importing File Resources). Specify the name of this resource here. Tagged value **resource** takes precedence over both parameters **name** and **data**. When a resource is specified, the service will only try to parse the file specified via the resource. Values that have been provided through parameters will be omitted. The resource can be replaced changing the resource path in the **Flat File Parser/Composer** settings of the xUML service. See xUML Service Settings for more information on changing the settings of a service.	any valid imported file resource added to the component diagram

Parameters

Action "compose"

Name	Type	Direction	Mandatory	Description	Allowed Values		Example
<any>	**<<FlatFile>>** class	in	✓	Provide an object containing the flat file data. The class defining the type of this object should have stereotype <<FlatFile>> and should depict the structure of the file.			
encoding	**String**	in		Provide the encoding of the file to be composed as specified in the Charset Definitions appendix.	any valid encoding (see Charset Definitions)		`UTF-8`
					default	`ISO-8859-1` (Latin1)	
locale	**NumbersLocale**	in		Specify how number values will be treated, when written to the flat file (decimal point, currency symbol, ...). You can overwrite the system locales here. Refer to Number Formatting for more information.	default	system locales	
name	**String**	in	(✓)	Specify a full path to the flat file, if you want to write the <<FlatFile>> object to the file system. Alternatively, you can compose the flat file to a **Blob** object (see parameter **data**). Note, that the **name** parameter takes priority over **data**.			`tmp/myFlatFile.txt`
data	**Blob**	out	(✓)	If you want to compose the <<FlatFile>> object to a **Blob** object, use this parameter as output of the compose action. Alternatively, you can write the composed flat file directly to the file system (see parameter **name**). Note, that the **name** parameter takes priority over **data**.			

If you provide both parameters, **name** and **data**, the <<FlatFile>> object will be written to the file system.

Action "parse"

Name	Type	Direction	Mandatory	Description	Allowed Values	Example
data	**Blob**	in	(✓)	Provide the flat file data to be parsed. Alternatively, you can specify a path to a flat file in the file system (see parameter **name**). Note, that the **name** parameter takes priority over **data**.		

name	**String**	in	(✓)	Specify a full path to the flat file to be parsed. Alternatively, you can parse the flat file from a **Blob** object (see parameter **data**). Note, that the **name** parameter takes priority over **data**.		`tmp/myFile.txt`
encoding	**String**	in		Provide the encoding of the file to be parsed as specified in the Charset Definitions appendix.	any valid encoding (see Charset Definitions) default: `ISO-8859-1` (Latin1)	`UTF-8`
locale	**NumbersLocale**	in		Specify how number values will be treated, when parsed from the flat file (decimal point, currency symbol, ...). You can overwrite the system locales here, if the file was written with divergent locales. Refer to Number Formatting for more information.		
<any>	**<<FlatFile>>** class	out	✓	The adapter returns a parsed flat file object. The class defining the type of this object should have stereotype <<FlatFile>> and should depict the structure of the file.		

If you provide both parameters, **name** and **data**, the <<FlatFile>> object will be parsed from the file system. If you specified tagged value **resource** (see section Tagged Values), the file will only be parsed from the resource.

Parameter Types

Class	Attribute	Type	Description
NumbersLocale	negativeSign	**String**	Characters used to signify negative values. Usually '-'.
	positiveSign	**String**	Characters used to signify positive values. Usually '+'.
	thousandsSeparator	**String**	Characters used to separate units of thousand, e.g. ','.
	decimalPoint	**String**	e.g. '.'
	currencySymbol	**String**	e.g. '$'