Scout Application Modeling Language

| **Document Version** | 0.14 |
| --- | --- |
| **Status** | Draft |
| **SAML Language Version** | >= 1.0.0.20130614 |
| **SAML Importer Version** | >= 3.10.0.20130614 |

| **Change History** | | | |
| --- | --- | --- | --- |
| **Change Log** | **Version** | **Author** | **Date** |
| Document created | 0.1 | Matthias Villiger | 2012-09-21 |
| Review and minor changes | 0.2 | Matthias Zimmermann | 2012-09-21 |
| Overview adapted & documentation started | 0.3 | Matthias Villiger | 2012-12-05 |
| SAML Definition added | 0.4 | Matthias Villiger | 2013-01-03 |
| SAML Documentation adapted for codes, forms & menus | 0.5 | Matthias Villiger | 2013-01-07 |
| Defaults for SAML attributes added | 0.6 | Matthias Villiger | 2013-01-08 |
| New Attributes for code element added | 0.7 | Matthias Villiger | 2013-01-11 |
| Check added, New ‘type’ attribute added for buttons, SAML importer architecture adapted | 0.8 | Matthias Villiger | 2013-02-14 |
| Filechooser, Radio Groups and Radio Options added | 0.9 | Matthias Villiger | 2013-02-15 |
| Adaptions for module element | 0.10 | Matthias Villiger | 2013-03-19 |
| Technical documentation for SAML Importer extended (more details about known problems, software architecture, headless run). | 0.11 | Matthias Villiger | 2013-05-31 |
| Chapter 4.2.1.1 added. | 0.12 | Matthias Villiger | 2013-06-06 |
| Chapter 4.3.1.4 added. | 0.13 | Matthias Villiger | 2013-06-07 |
| Format Attribute added, Documentation for Long Field added | 0.14 | Matthias Villiger | 2013-06-14 |

**Table of Contents**

[1 Abstract 5](#_Toc358978822)

[2 Introduction 6](#_Toc358978823)

[3 Workflow Overview 7](#_Toc358978824)

[3.1 Parser 7](#_Toc358978825)

[3.2 SAML Emitter 7](#_Toc358978826)

[3.3 SAML 7](#_Toc358978827)

[3.4 SAML Importer 7](#_Toc358978828)

[3.5 Scout SDK API 8](#_Toc358978829)

[4 Documentation 9](#_Toc358978830)

[4.1 Overview 9](#_Toc358978831)

[4.2 SAML Definition 10](#_Toc358978832)

[4.2.1 General 10](#_Toc358978833)

[4.2.1.1 Escaping of Keywords 10](#_Toc358978834)

[4.2.2 Module 10](#_Toc358978835)

[4.2.3 Translation 11](#_Toc358978836)

[4.2.4 Code 11](#_Toc358978837)

[4.2.5 Lookup 12](#_Toc358978838)

[4.2.6 Template 12](#_Toc358978839)

[4.2.7 Form 13](#_Toc358978840)

[4.2.7.1 Form Field 14](#_Toc358978841)

[4.2.7.1.1 Sequence 14](#_Toc358978842)

[4.2.7.1.2 Group 15](#_Toc358978843)

[4.2.7.1.3 String 15](#_Toc358978844)

[4.2.7.1.4 BigDecimal 16](#_Toc358978845)

[4.2.7.1.5 Long 18](#_Toc358978846)

[4.2.7.1.6 Date 19](#_Toc358978847)

[4.2.7.1.7 Smartfield 19](#_Toc358978848)

[4.2.7.1.8 List 20](#_Toc358978849)

[4.2.7.1.9 Custom Field 21](#_Toc358978850)

[4.2.7.1.10 Button 22](#_Toc358978851)

[4.2.7.1.11 Check 23](#_Toc358978852)

[4.2.7.1.12 Table 24](#_Toc358978853)

[4.2.7.1.13 Tabbox 25](#_Toc358978854)

[4.2.7.1.14 File Chooser 26](#_Toc358978855)

[4.2.7.1.15 Radio Group 27](#_Toc358978856)

[4.2.7.2 Key Stroke 29](#_Toc358978857)

[4.2.7.3 Menu 29](#_Toc358978858)

[4.2.8 Logic 30](#_Toc358978859)

[4.3 SAML Importer 32](#_Toc358978860)

[4.3.1 Architecture 33](#_Toc358978861)

[4.3.1.1 Element Importer 33](#_Toc358978862)

[4.3.1.1.1 Form Field Element Importer 34](#_Toc358978863)

[4.3.1.2 Attribute Handlers 35](#_Toc358978864)

[4.3.1.3 SAML Context 35](#_Toc358978865)

[4.3.1.4 Entry Points 36](#_Toc358978866)

[4.3.2 Generated Artifacts 37](#_Toc358978867)

[4.3.2.1 G01.01 Client Service Interface 38](#_Toc358978868)

[4.3.2.2 G01.02 Client Service 38](#_Toc358978869)

[4.3.2.3 G01.03 Process Service Interface 39](#_Toc358978870)

[4.3.2.4 G01.04 Process Service 39](#_Toc358978871)

[4.3.2.5 G01.05 Code Type 40](#_Toc358978872)

[4.3.2.6 G01.06 Form 40](#_Toc358978873)

[4.3.2.7 G01.07 Form Data 40](#_Toc358978874)

[4.3.2.8 G01.08 Lookup Service Interface 41](#_Toc358978875)

[4.3.2.9 G01.09 Lookup Service 41](#_Toc358978876)

[4.3.2.10 G01.10 Lookup Call 42](#_Toc358978877)

[4.3.2.11 G01.11 Texts.properties 42](#_Toc358978878)

[4.3.2.12 G01.12 Text Provider Service 42](#_Toc358978879)

[4.3.2.13 G01.13 Preferences 43](#_Toc358978880)

[4.3.3 Headless Run 43](#_Toc358978881)

[4.3.4 Known Problems 43](#_Toc358978882)

[4.3.4.1 Super class of <type> could not be found 43](#_Toc358978883)

[4.3.4.2 Unable to load secure storage module (<module name>) 44](#_Toc358978884)

[4.3.4.3 IllegalArgumentException 44](#_Toc358978885)

[4.3.4.4 SWTError: No more handles […] 44](#_Toc358978886)

[5 List of Abbreviations 45](#_Toc358978887)

[6 Referenced Documents 46](#_Toc358978888)

# Abstract

This document describes the motivation for the definition of the Scout Application Modeling Language (SAML). We also provide an example workflow explaining how Scout application code may be generated based on SAML files.

The supported SAML language elements are outlined and explained.

Furthermore the inputs and generated artifacts of the workflow are listed. Finally a technical overview over the software architecture of the SAML importer is given.

# Introduction

Eclipse Scout (<http://www.eclipse.org/scout/>) is a framework for creating modern business applications. Such applications may be separated into multiple tiers where each tier is responsible for a specific part of the application. Examples for tier responsibilities are presenting information to the user, processing business logic or persisting data. The goal of Eclipse Scout is maximizing the efficiency of building distributed enterprise applications based on the Eclipse platform without losing flexibility.

To further simplify the creation of Scout applications and to give third party software a simple and compact language to define Scout applications, a new file format is to be designed. The intention is to hide any Java or Eclipse specific elements. For this purpose the Scout Application Modeling Language (SAML) has been created. Examples can be found in 1.

As a first use of SAML a customer who uses Scout wants to migrate existing legacy applications written in different languages into a new, Scout based Java application. For this, SAML is used to migrate the old applications into the new one. Advantages of this approach are expected in two areas. First, due to SAML no details about the Scout framework concepts or architecture are necessary. Second, SAML output generated during the migration remains valid while the Scout framework API is allowed to further evolve.

To ensure the generated code fulfills all Scout conventions and is fully compatible with the Eclipse Scout Tooling the existing Scout SDK Code Generation API is to be reused. This API includes operations and tools to generate Scout projects and source code. The same API is also used in the graphical Scout Tooling which is integrated into the Eclipse IDE.

Xtext (<http://www.eclipse.org/Xtext/>) should be used to parse, validate, read and write SAML files and provide some tooling support. Furthermore it should be possible to transfer the SAML files into a memory model so that a code generator can write the corresponding Scout code using the Scout SDK Code Generation API.

The current focus of the described effort concentrates on the definition of the SAML. Important requirements are excellent human readability, simplicity to support the learning process for the language, modularity, and flexibility for future enhancements. For the first project we expect that the complete application will be described in a couple of thousands of SAML files.

# Workflow Overview

This section gives an overview over a possible migration workflow using SAML and the components involved. Below the individual components are described in more detail.



## Parser

The parser is a software tool to parse, analyze, and transform existing software source code written in various languages. It is capable to build an Inventory containing all information about the original application. Depending on the requirements applying various code analyses, transformation and modification routines may be necessary.

## SAML Emitter

The SAML Emitter is a module that is capable to generate SAML source code based on the Inventory model. Emitters for Java may also be necessary to use the Java code snippet support included in SAML. The SAML language already included an API to generate SAML source code.

## SAML

The SAML is defined using the Eclipse Xtext framework. Based on this definition (see 2) the following artifacts are (partly) generated:

* Validator/Parser/Writer: Code that checks a SAML file for correctness and completeness and can load the content of a file it into a memory representation (EMF model) or write it to files.
* SAML Editor: A text editor for SAML files including syntax highlighting, content assist, validation & quick fixes.
* SAML Model: EMF model that represent a SAML file.

## SAML Importer

The importer traverses over the SAML model. The model classes have been generated by the SAML definition using Xtext. At runtime, this model is filled using the SAML files. The importer then creates Scout code using the existing Scout SDK API.

## Scout SDK API

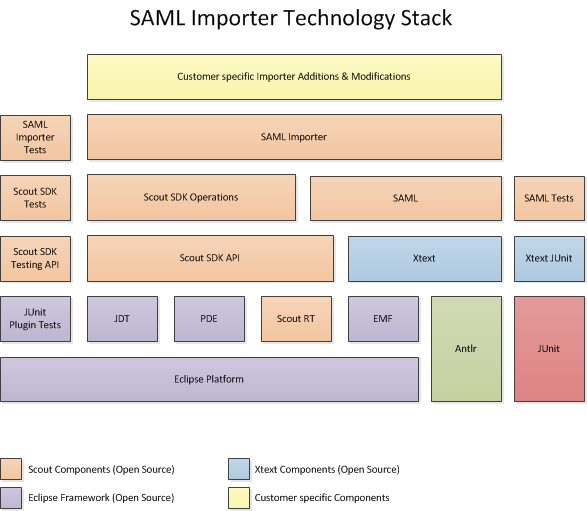
The Scout SDK already includes powerful tools to generate Scout source code. This API should be reused to write the final Scout source code. This ensures that the generated code is complete and respects all Scout conventions. Furthermore the generated code will then be completely compatible with the Scout SDK UI integration into Eclipse which uses the same API.

# Documentation

This section contains the technical documentation of the SAML language and the SAML Importer.

## Overview

The following diagram illustrates the technology stack of the SAML and the corresponding Importer.



## SAML Definition

This section describes the SAML language elements. Each element is discussed and examples are given. Furthermore all its attributes and their type restrictions are listed and the possible child objects are described.

### General

This section contains general conventions and rules that apply for all SAML elements.

#### Escaping of Keywords

SAML uses some keywords as identifiers. All keywords are described in the next sections. If you want to use a name for an element which also exists as SAML keyword, you have to escape the reserved part. The escaping character (as typically used by Xtext) is “^”.

Example:

**module** a.test.^shared

**translation** My.^form.test.Name en="en" de="de" de\_CH="de\_CH"

**form** ^form {

}

### Module

A module must be specified at the beginning of each SAML file. It defines in which Scout project (or sub-project) the contents of the file belong to. The module name must match a Scout project in the workspace and must be a valid qualified name.

Example:

**module** org.eclipse.scout.helloworld

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | Qualified Name |  | Specifies in which plugins the generated source code will be placed. E.g. when “org.eclipse.scout” is provided as name, the source will be emitted into plugins named “org.eclipse.scout.client”, “org.eclipse.scout.shared” or “org.eclipse.scout.server”. These plugins must be available in the workspace. |
| client | Qualified Name |  | If a different client should be used than the one used by default (see attribute “name”), the symbolic name of the plugin can be specified here. It must be a bundle of type client and it must have the shared plugin on its classpath. |
| shared | Qualified Name |  | If a different shared should be used than the one used by default (see attribute “name”), the symbolic name of the plugin can be specified here. It must be a bundle of type shared. |
| server | Qualified Name |  | If a different server should be used than the one used by default (see attribute “name”), the symbolic name of the plugin can be specified here. It must be a bundle of type server and it must have the shared plugin on its classpath. |

### Translation

A translation represents a text to display to the user. Such a translation is identified by an application wide unique key. Each translation contains texts in different languages.

Example:

**translation** MyApp.Labels.Msg default\_lang="Message" de="Nachricht" it\_CH="Messaggio" […]

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | Qualified Name |  | Unique Id over all SAML files in the same scope. |
| language | ID |  | One of the following options:   * ISO 639 alpha-2 language code * ISO 639 alpha-2 language code & “\_” & ISO 3166 alpha-2 country code * “default\_lang”   The „default\_lang“ specifies the text to be used when no text for a specific language is present. E.g. if you have a system locale set to Spanish but your application is only translated to English and French, then one of those two should be the default language that will be used when Spanish is not available. |
| text | String |  | The text translated into the given language. |

### Code

A code is a structure to represent a tree key-code association

See <http://wiki.eclipse.org/Scout/Concepts/CodeType> for more details about codes.

Example:

**code** Parent **id**="1234" {

**code** Child1 **id**="\"child1\"" **value\_type**="java.lang.String"

**code** Child2 **id**="\"child2\"" {

**code** Child2\_1 **id**="1234" **value\_type**="java.lang.Integer"

}

**code** Child3 **id**="new CodeType()" **value\_type**="aa.shared.codes.CodeType" **super\_type**=CodeTypeAbs

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| id | String |  |  | The value of the code.  Note: this value is directly inserted into the java code. This allows to also add e.g. number literals. But if you want to use string, you must add the starting and ending quotes and care about escaping (see example above). |
| value\_type | String |  | java.lang.Object | Fully qualified name to the class that defines the type of value for this code.  The value type of a code is typically the type of the id of this code.  Can also be a simple name if the class name is unique on the classpath. |
| text | Translation |  |  | Reference to the name of a translation element. |
| super\_type | Template |  | AbstractCode or AbstractCodeType | Reference to the name of a template element.  The default used depends on the element location: Top level elements use AbstractCodeType, inner types use AbstractCode |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed inside this code. See chapter 4.2.7 for more details about logic elements. |
| codes |  | Inner codes. |

### Lookup

A lookup element represents a key-value-pair data source for e.g. listboxes or smartfields. A lookup is backed by a backend service providing the data to the client.

See <http://wiki.eclipse.org/Scout/Concepts/LookupCall> for more details about lookups.

Example:

**lookup** Departments {

**logic** **placement**=**server** {

" private LookupRow[] createDummyData() {

return new LookupRow[]{new LookupRow(100L, \"Value 1\"), new LookupRow(200L, \"Value 2\")};

}

"

}

**logic** **event**=**all** **placement**=**server** {

"return createDummyData();"

}

}

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | ID |  |  |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the service providing the data for this lookup. See chapter 4.2.7 for more details about logic elements. |

### Template

A template is a class that encapsulates some very often used components. Instead of always repeating the same Scout configurations again, a template can be created and then reused.

See <http://wiki.eclipse.org/Scout/Concepts/Template> for more details about templates.

Example:

**template M**sgField **definition=**"org.eclipse.scout.helloworld.client.ui.templates.AbstractMessageField"

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | ID |  |  |
| definition | String |  | Fully Qualified Name to the class of the template. |

### Form

A form is a UI dialog containing form field widgets, validation and event-handling (calling client- or server-logic services).

See <http://wiki.eclipse.org/Scout/Concepts/Form> for more details about forms and <http://wiki.eclipse.org/Scout/Examples/SimpleForms> for an example how a simple form could look like.

Example:

**form** Test **modal**=**true title**=FormTitle **columns**=1 {

[…]

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| modal | Boolean |  | true |  |
| columns | Integer > 0 |  | 2 | Specifies how many columns the layout grid of the form should have. |
| title | Translation |  | no title | Reference to the name of a translation element. |
| subtitle | Translation |  | no subtitle | Reference to the name of a translation element. |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractForm | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| fields |  | The form fields of this form.  See below for a list and more details about form fields. |
| keyStrokes |  | The keystrokes events for this form.  See below for more details about key strokes. |

#### Form Field

Form fields are widgets shown in a form. See <http://wiki.eclipse.org/Scout/Concepts/Field> for more details about the different form fields that exist (not all Scout form fields may already be supported by SAML).

The table below gives an overview over all form fields. Afterwards each field is documented.

| **Form Field** | **Composite** | **Details Link** |
| --- | --- | --- |
| bigdecimal | n | <http://wiki.eclipse.org/Scout/Concepts/BigDecimalField> |
| date | n | <http://wiki.eclipse.org/Scout/Concepts/DateField> |
| smartfield | n | <http://wiki.eclipse.org/Scout/Concepts/SmartField> |
| list | n | <http://wiki.eclipse.org/Scout/Concepts/ListBox> |
| custom\_field | n |  |
| button | n | <http://wiki.eclipse.org/Scout/Concepts/Button> |
| table | n | <http://wiki.eclipse.org/Scout/Concepts/TableField> |
| tabbox | n | <http://wiki.eclipse.org/Scout/Concepts/TabBox> |

##### Sequence

A sequence is a composite field used mainly for layouting.

See <http://wiki.eclipse.org/Scout/Concepts/SequenceBox> for more details.

Example:

**sequence** MySeq **text**=MyLabel {

[…]

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  |  |  |
| super\_type | Template |  | AbstractSequenceBox | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| fields |  | The form fields that will be placed inside this composite. |

##### Group

A group is a composite field used for grouping and layouting.

See <http://wiki.eclipse.org/Scout/Concepts/GroupBox> for more details.

Example:

**group** Grp02 **columns**=1 **width**=1 **border\_decoration**=**line border\_visible**=**true** {

[…]

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  |  | Reference to the name of a translation element. |
| columns | Integer > 0 |  | 2 | Specifies how many columns the layout grid of the group should have. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| border\_visible | Boolean |  | true |  |
| border\_decoration | Enum |  | auto | One of   * empty * line * section * auto |
| height | Integer > 0 |  | content defined |  |
| width | Integer > 0 |  | full width |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractGroupBox | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| fields |  | The form fields that will be placed inside this composite. |

##### String

A string is a form field to enter text.

See <http://wiki.eclipse.org/Scout/Concepts/StringField> for more details.

Example:

**string** ZregKurzbezeichnung **enabled**=**false label\_visible**=**false**

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| maxlen | Integer > 0 |  | 4000 | Specifies the maximum number of characters that can be entered. |
| height | Integer > 0 |  | 1 | Specifies the height of the string field. If the height is > 1 the string field is configured to be multi line (text area). |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| horizontal\_align | Enum |  | left | One of:   * left * center * right |
| super\_type | Template |  | AbstractStringField | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### BigDecimal

A bigdecimal is a field to enter arbitrary-precision signed decimal numbers.

See <http://wiki.eclipse.org/Scout/Concepts/BigDecimalField> for more details.

Example:

**bigdecimal** Currency01 **fraction\_digits**=4 **min**=0 **max**=20000.5 **grouping**=**true** **horizontal\_align**=**left**

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| percent | Boolean |  | false | If true, the field will automatically include a percent sign (“%”) at the end of the number. |
| grouping | Boolean |  | true | If true, the field will automatically group the number. The grouping separator as well as the size of each group is locale dependent.  Cannot be used if the “format” attribute is provided. |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| horizontal\_align | Enum |  | right | One of:   * left * center * right |
| fraction\_digits | Integer > 0 |  | 2 | Specifies the maximum number of digits after the decimal separator.  Cannot be used if the “format” attribute is provided. |
| min | Decimal |  | unlimited | Specifies the minimal value that will be allowed. |
| max | Decimal |  | unlimited | Specifies the maximum value that will be allowed. |
| format | String |  | none | Allows specifying a number format. Values entered into the field will be formatted using the given pattern.  The attributes “grouping” and “fraction\_digits” cannot be used then because these information is then provided in the given format.  See the [DecimalFormat Doc](http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html) for possible patterns. |
| super\_type | Template |  | AbstractBig DecimalField | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### Long

A long is a field that holds numbers without decimal places.

See <http://wiki.eclipse.org/Scout/Concepts/LongField> for more details.

Example:

**long** NumElements **min**=1 **max**=9999 **format**="0000"

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| horizontal\_align | Enum |  | right | One of:   * left * center * right |
| min | Long |  | unlimited | Specifies the minimal value that will be allowed. |
| max | Long |  | unlimited | Specifies the maximum value that will be allowed. |
| format | String |  | none | Allows specifying a number format. Values entered into the field will be formatted using the given pattern.  See the [DecimalFormat Doc](http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html) for possible patterns. |
| super\_type | Template |  | AbstractLongField | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### Date

A date is a field to enter a date in the Gregorian calendar. The field supports date-picking using a calendar popup.

See <http://wiki.eclipse.org/Scout/Concepts/DateField> for more details.

Example:

**date** MyDate **label\_visible**=**false**

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| format | String |  | none | Allows specifying a date format. Dates entered into the field will be formatted using the given pattern.  See the [SimpleDateFormat Doc](http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html) for possible patterns. |
| super\_type | Template |  | AbstractDate Field | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### Smartfield

A smartfield is a combo-box like field supporting “search-as-you-type”-behavior.

See <http://wiki.eclipse.org/Scout/Concepts/SmartField> for more details.

Example:

**smartfield** MySmartField **text**=MyLabel **enabled**=**false** **code**=MyTest **value\_type**="java.lang.Integer"

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| code | Code |  | none | Reference to a code element to use as data source for this smartfield. See chapter 4.2.3 for details about codes. |
| value\_type | String |  | java.lang.Object | Fully qualified name to the class that defines the type of value for this smartfield.  The value type of a smartfield is typically the same type as the id of a code (when codes are used) or the same type as the key column of a lookup row (when lookups are used).  Can also be a simple name if the class name is unique on the classpath. |
| lookup | Lookup |  | none | Reference to a lookup element to use as data source for this smartfield. See chapter 4.2.4 for details about lookups. |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractSmart Field | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| menus |  | Context menus that will be available on this smartfield. See chapter 4.2.6.3 for more details about context menus. |

##### List

A list is a box that allows selecting multiple values from a list.

See <http://wiki.eclipse.org/Scout/Concepts/ListBox> for more details.

Example:

**list** MyList **text**=MyLabel **lookup**=MyData **value\_type**="org.eclipse.scout.helloworld.shared.beans.MyBean"

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| code | Code |  | none | Reference to a code element to use as data source for this list. See chapter 4.2.3 for details about codes. |
| value\_type | String |  | java.lang.Object | Fully qualified name to the class that defines the type of value for this list.  The value type of a list is typically the same type as the id of a code (when codes are used) or the same type as the key column of a lookup row (when lookups are used).  Can also be a simple name if the class name is unique on the classpath. |
| lookup | Lookup |  | none | Reference to a lookup element to use as data source for this list. See chapter 4.2.4 for details about lookups. |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractListBox | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### Custom Field

A custom field represents a field that does not exist in the Scout runtime.

Example:

**custom\_field** MyCustom **super\_type**=MyTestField **text**=MyLabel

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | ID |  |  |
| super\_type | Template |  | Reference to the name of a template element. |
| text | Translation |  | Reference to the name of a translation element. |
| enabled | Boolean |  |  |
| visible | Boolean |  |  |
| label\_visible | Boolean |  |  |
| width | Integer > 0 |  |  |
| width\_pixels | Integer > 0 |  |  |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| fields |  | The form fields that will be placed inside this composite. |

##### Button

A button that invokes an action when clicked by the user.

See <http://wiki.eclipse.org/Scout/Concepts/Button> for more details.

Example:

**button** OpenOtherForm **text**=OpenPopupLabel

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | none | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| process\_button | Boolean |  | true | Configures whether this button is a process button. Process buttons are typically displayed on a dedicated button bar at the bottom of a form. Non-process buttons can be placed anywhere on a form. |
| type | Enum |  | normal | A special button type. Depending on this type the button has an implicit behavior. One of:   * normal no default behavior. * cancel closes the form. If the user changed anything in the form she/he is asked if it should be saved. If yes, the form behaves as if an Ok button would have been pressed. Otherwise it is directly closed. * close Directly closes the form without any message or without storing it. * ok Directly closes the form and automatically saves the form if the user changed anything. * reset Sets all fields in the form back to their initial value. The form remains open. * save Saves the form if the user changed anything. The form remains open. |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractButton | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| menus |  | Context menus that will be available on this button. See chapter 4.2.6.3 for more details about context menus. |

##### Check

A checkbox that can only have two states: active (true) or inactive (false).

See <http://wiki.eclipse.org/Scout/Concepts/CheckboxField> for more details.

Example:

**check** Test **text**=trans.text2 **mandatory**=**true**

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| super\_type | Template |  | AbstractButton | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### Table

A table is a field showing a table containing columns and menus.

See <http://wiki.eclipse.org/Scout/Concepts/TableField> for more details.

Example:

**table** Test **label\_visible**=**false** **height**=10 **width**=2 {

**menu** Edit **text**=LabelEdit

**column** ColId **type**=**int** **width**=0 **displayable**=**false**

**column** ColText **text**=Text **type**=**string** **width**=138

**column** ColValuta **text**=Valuta **type**=**date** **width**=100 **visible**=**false**

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| label\_visible | Boolean |  | true |  |
| height |  |  | 1 |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractTableField | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| menus |  | Context menus that will be available on this smartfield. See chapter 4.2.6.3 for more details about context menus. |
| columns |  | The columns of the table. See chapter 4.2.6.4 for more details about columns. |

###### Column

A column describes a single part of a table structure.

See <http://wiki.eclipse.org/Scout/Concepts/Column> for more details about menus.

Example:

**column** ColId **type**=**int** **width**=0 **displayable**=**false**

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| type | Enum |  |  | One of:   * date * string * int * bigdecimal |
| visible | Boolean |  | true | Specifies if the column is shown by default. |
| text | Translation |  | no text | Reference to the name of a translation element. Specifies the header text of the column. |
| width | Integer > 0 |  | 60 |  |
| displayable | Boolean |  | true | Specifies if the column can be made visible by the user. |

##### Tabbox

The tabbox is a field to create tabs containing other fields.

See <http://wiki.eclipse.org/Scout/Concepts/TabBox> for more details.

Example:

**tabbox** MyTab {

**tab** Detail **text**=Tab01Text {

[…]

}

**tab** Documents **text**=Tab02Text {

[…]

}

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| width | Integer > 0 |  | full width |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| super\_type | Template |  | AbstractTableField | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| tabs |  | The tabs that will be available. See chapter 4.2.6.5 for more details about tabs. |

###### Tab

A tab is a composite used inside a tabbox.

See <http://wiki.eclipse.org/Scout/Concepts/TabBox> for more details.

Example:

**tab** Detail **text**=Tab01Text {

[…]

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| columns | Boolean |  | 2 | Specifies how many columns the layout grid of the tab should have. |
| super\_type | Template |  | AbstractGroupBox | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| fields |  | The form fields that will be placed inside this composite. |

##### File Chooser

A file chooser is a field to select a file or directory on the local hard-drive.

See <http://wiki.eclipse.org/Scout/Concepts/FileChooserField> for more details.

Example:

**filechooser** TestFileChooser **text**=trans.text2 **mode**=**file** **type**=**store** **extensions**="jpg jpeg png bmp gif"  
 **default\_directory**="C:\\Program Files" **show\_directory**=**true** **show\_filename**=**true** **show\_extension**=**true**

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| maxlen | Integer > 0 |  | 4000 | Specifies the maximum number of characters that can be entered. |
| mode | Enum |  | file | Specifies if the file chooser should allow to select files or directories.  One of:   * file * directory |
| type | Enum |  | store | Specifies if the file chooser should behave as if a file/directory should be stored or loaded.  One of:   * load * store |
| extensions | String |  | all (\*) | Only if mode=file  Specifies which file extensions should be shown in the browse dialog. The extensions are separated by space, comma or semicolon (see example above). |
| default\_directory | String |  | depends on OS | Specifies the default directory where the browse dialog should open. |
| show\_directory | Boolean |  | false | After a file/directory has been selected: specifies if the folder path is displayed in the field. |
| show\_filename | Boolean |  | true | After a file has been selected: specifies if the filename is displayed in the field. |
| show\_extension | Boolean |  | true | After a file has been selected: specifies if the file extension is displayed in the field. |
| super\_type | Template |  | AbstractButton | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

##### Radio Group

A radio group is a specific type of value field to propose a choice between a limited numbers of possibilities.

See <http://wiki.eclipse.org/Scout/Concepts/RadioButtonGroup> for more details.

Example:

**radio\_group** TestRadioButtons **text**=trans.text1 **mandatory**=**true** **value\_type**="java.lang.Integer" {

**option** Opt1 **text**=trans.text1 **value**="10"

**option** Opt2 **text**=trans.text2 **value**="20"

**option** Opt3 **text**=trans.text3 **value**="30"

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| width | Integer > 0 |  | 1 |  |
| width\_pixels | Integer > 0 |  | undefined |  |
| label\_visible | Boolean |  | true |  |
| master | Form Field |  | none | Reference to a form field inside the same form having one of the following types:   * string * bigdecimal * long * date * smartfield * list * check * radio\_group * filechooser |
| mandatory | Boolean |  | false |  |
| value\_type | String |  | java.lang.Object | Fully qualified name to the class that defines the type of value for this radio group.  The value type of a radio group is typically the same data type as used in the value attribute of the child options (see 4.2.6.1.14.1). |
| super\_type | Template |  | AbstractButton | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| options |  | Options that can be selected for this group. See chapter 4.2.6.1.14.1 for more details about options. |

###### Option

An option is a single value that can be selected inside a radio group. If one option is selected, the one selected before is automatically unchecked.

See <http://wiki.eclipse.org/Scout/Concepts/RadioButton> for more details.

Example:

**option** Opt **text**=labels.Options.TestOpt **value**="30"

**option** OptWithStringValue **text**=labels.Options.TestOpt2 **value**="\"string value\""

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| value | String |  | null | Specifies the value that belongs to the current option. If this option is selected, the surrounding radio group returns this value.  Note: the value is directly inserted into the java source code. This allows using e.g. number literals. If you want to return a string value you must add the starting and ending quotes and you have to care about escaping (see example above). |
| super\_type | Template |  | AbstractRadioButton | Reference to the name of a template element. |

#### Key Stroke

A key stroke is a key press event binding. Key strokes can only be defined in a form on top level.

See <http://wiki.eclipse.org/Scout/Concepts/KeyStroke> for more details about key strokes.

Example:

**key** MyStroke **stroke**="f6" {

**logic** **event**=**activated** **placement**=**server** {

"[…]"

}

}

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | ID |  |  |
| stroke | String |  | The key on which the nested actions should be executed. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |

#### Menu

A menu is tree like structure that invokes actions on a mouse click.

See <http://wiki.eclipse.org/Scout/Concepts/Menu> for more details about menus.

Example:

**menu** Exit **text**=Labels.Exit {

**logic** **event**=**click** **placement**=**client** {

"[…]"

}

}

| **Attribute** | **Type** | **Mandatory** | **Default** | **Comment** |
| --- | --- | --- | --- | --- |
| name | ID |  |  |  |
| enabled | Boolean |  | true |  |
| visible | Boolean |  | true |  |
| text | Translation |  | no text | Reference to the name of a translation element. |
| single\_select | Boolean |  | true | Specifies if the menu should be available when one item is selected. |
| multi\_select | Boolean |  | false | Specifies if the menu should be available when more than one item is selected. |
| empty\_select | Boolean |  | false | Specifies if the menu should be available when no item is selected. |
| type | Enum |  | normal | One of:   * normal (normal context menu) * separator (this item will be displayed as separator: horizontal line) * cancel (this menu will automatically close the current form. behaves as if the cancel button would have been clicked) |
| super\_type | Template |  | AbstractMenu | Reference to the name of a template element. |

| **Children** | **Mandatory** | **Comment** |
| --- | --- | --- |
| logic |  | Logic snippets that will be placed in the client- or server-service of this form. See chapter 4.2.7 for more details about logic elements. |
| menus |  | Context menus that will be available below this menu. See chapter 4.2.6.3 for more details about context menus. |

### Logic

A logic element is used to wire java business logic into the Scout elements. A logic element can be named or anonymous.

A named logic snippet can be defined if the source code should be used at several places. This allows defining the Java code once and referring to the logic from various places (using the exec attribute).

An anonymous logic snippet must be placed inside the element where it should be used and cannot be re-used.

Named example:

**logic** MyLogicSnippet **placement**=**server** {

"[…]"

}

**form** MyForm **{**

**string** Str01{

**logic event**=**activated exec**=MyLogicSnippet

}

}

Anonymous example:

**form** MyForm **{**

**string** Str01{

**logic placement**=**server event=activated** {

"[…]"

}

}

}

| **Attribute** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| name | ID |  |  |
| event | Enum |  | One of:   * all * modify\_load * modify\_store * modify\_discard * new\_load * new\_store * new\_discard * changed * click * master\_changed * init * format\_value * activated * loaded |
| placement | Enum |  | One of:   * client The logic is placed in the client service corresponding to the surrounding form. * Server The logic is placed in the server service corresponding to the surrounding form. * Inline The logic is added directly on top level of the surrounding class. |
| exec | Logic |  | Reference to a named logic element. If this attribute is given, the current logic element cannot have own source. |

| **Child** | **Type** | **Mandatory** | **Comment** |
| --- | --- | --- | --- |
| source | String |  | The Java source code that will be inserted in the corresponding class. Cannot be provided if the exec attribute is specified. |

The event that can be used inside an element depends on the element type. The following table explains which events exist for which surrounding types.

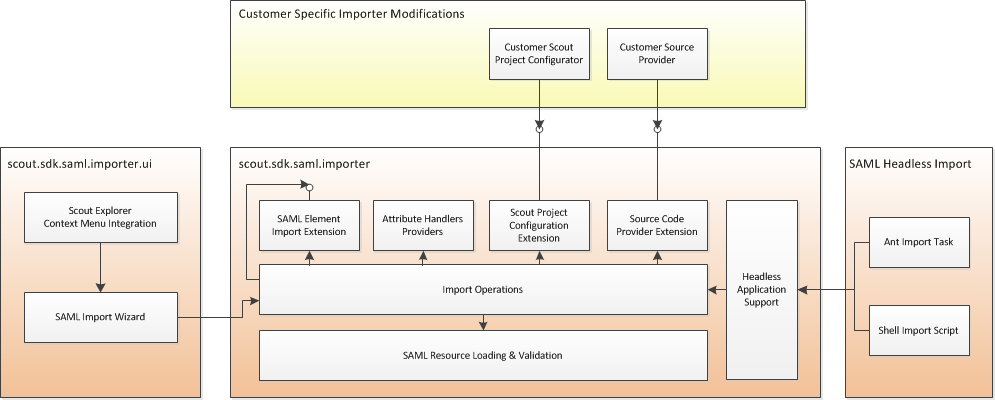
| **Surrounding Type** | **Events** |
| --- | --- |
| Form | * modify\_load * modify\_store * modify\_discard * new\_load * new\_store * new\_discard |
| Button | * click * init |
| Menu | * click |
| Lookup | * all |
| Table | * init |
| Key Stroke | * activated |
| String | * changed * format\_value * master\_changed * init |
| BigDecimal | * changed * format\_value * master\_changed * init |
| Long | * changed * format\_value * master\_changed * init |
| Date | * changed * format\_value * master\_changed * init |
| Smartfield | * changed * format\_value * master\_changed * init |
| List | * changed * format\_value * master\_changed * init |
| Check | * changed * format\_value * master\_changed * init |
| Radio Group | * changed * format\_value * master\_changed * init |
| File Chooser | * changed * format\_value * master\_changed * init |
| Tabbox | * init |
| Custom Field | * init |
| Sequence | * init |
| Group | * init |
| Code | * load |

## SAML Importer

This section describes the technical architecture of the Importer. The generated output artifacts are explained based on sample input.

### Architecture

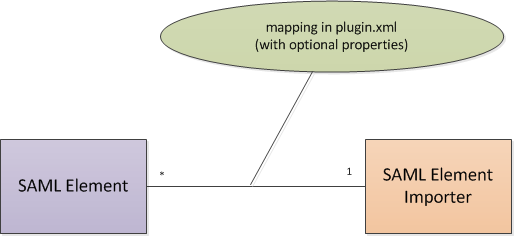
The following diagram gives a more detailed view on the architecture of the importer. It covers the Importer core components (orange) and customer specific adaptions (yellow).



* **Importer UI**  
  This module contains the integration into the graphical Eclipse Integration of Scout (<http://wiki.eclipse.org/Scout/SDK>). It provides context menus and wizards to import SAML files.
* **Importer**  
  This module contains the entire import logic. It is responsible for loading and validating the given SAML input and contains the import operations that create the necessary artifacts and the attribute handler providers that create the element configuration. Furthermore it provides extension points for customer specific enhancements in the area of scout project configurations, generated source code customization and import operations. Finally it contains the import application used by the headless import task.
* **Headless Import**  
  This module is a collection of scripts to run the import operation on command line without starting a graphical workbench. It contains shell scripts and ant macros to trigger a headless SAML import.
* **Customer Specific Modifications**  
  This module is optional. If existing, it can change, enhance or replace the generated source code with custom specific snippets. The importer then uses the provided source code when connecting the different generated artifacts (see chapter 4.3.2).

#### Element Importer

An Element importer is responsible to create the source object for a SAML element. For each element in a SAML file (e.g. a **form** Element) the corresponding handler is invoked.

The mapping between the SAML element (more detailed its Java interface) and the element importer that takes care of it is defined in the plugin.xml under the org. eclipse.scout.sdk.saml.importer.elementImporters extension point.

The order in which the elements are imported is given by the pipeline defined in org.eclipse.scout.sdk.saml.importer.operation.SamlImportOperation. This is required because the elements have inner dependencies. So for instance all **translation** elements must be imported as first step because they may be used by elements coming later on (e.g. **forms**). For this the import first traverses over all elements of a certain type in all SAML files before going on to the next element type.

##### Form Field Element Importer

Element importers can be configured in the plugin.xml where they are registered. This allows e.g. to use one element importer for several form fields by mapping always the same importer class (with different configurations) to different SAML elements. Here is an example:

<importer element="org.eclipse.scout.saml.saml.SequenceBoxElement"

operation=" org.eclipse.scout.sdk.saml.importer...FormFieldElementImportOperation">

<property name="fieldSuffix" value="Box" />

<property name="defaultSuperInterfaceFqn" value="org.eclipse.scout.rt...ISequenceBox" />

</importer>

<importer element="org.eclipse.scout.saml.saml.StringElement"

operation=" org.eclipse.scout.sdk.saml.importer...FormFieldElementImportOperation">

<property name="fieldSuffix" value="Field" />

<property name="defaultSuperInterfaceFqn" value="org.eclipse.scout.rt...IStringField" />

</importer>

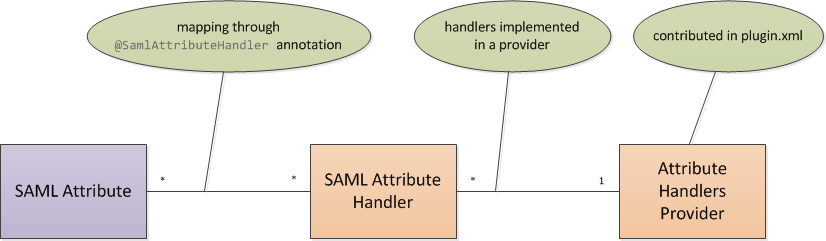
This example uses the same importer (FormFieldElementImportOperation) for two SAML elements: SequenceBoxElement and StringElement. But the import operation is configured differently. The first is creating a form field implementing ISequenceBox using “Box” as suffix while the second creates an IStringField using “Field” as suffix.

Which concrete ISequenceBox or IStringField implementation that will be used is decided according to the following list:

1. If existing: the explicit **super\_type** attribute specified on the current SAML element.
2. If existing: the project specific super class as defined in the Scout SDK preferences (see 4.3.2.13).
3. If org.eclipse.scout.rt.extension.client is on the classpath of the current bundle and there exists an org.eclipse.scout.rt.extension.client.IExtensibleScoutObject for the current SAML element: The extensible Scout class.
4. The default Scout class.

#### Attribute Handlers

Attribute handlers are methods responsible for applying a SAML attribute to the source object created by an element importer (see 4.3.1.1). So if an element importer has created e.g. a **form** source object (class), all the attribute handlers registered for the SAML attributes of that form are invoked.



One attribute handler can process several different attributes of several different SAML elements. The possible attributes that can be processed are defined using the @SamlAttributeHandler annotation. Attribute handlers are bundled into providers which themselves are registered in the plugin.xml file. See also the following example:

**public** **class** ExampleAttributeHandlersProvider **extends** AbstractAttributeHandlersProvider {

@SamlAttributeHandler(handles = {

@SamlAttribute(elementType=MySamlElement.**class**, featureId=SamlPackage.*MY\_SAML\_ELEMENT\_\_ATTRIB*),

@SamlAttribute(elementType=MyOtherElement.**class**, featureId=SamlPackage.*MY\_OTHER\_ELEMENT\_\_ATTRIB2*)

})

**public** **void** applyCodeAttribute(TypeOfAttrib a) **throws** CoreException, IllegalArgumentException {

…

}

}

This example attributes provider contains one SAML attribute handler that is capable to process two attributes (**attrib** on element MySamlElement and **attrib2** on element MyOtherElement).

Notes:

* All the attributes that one handler belongs to must be of the same type (TypeOfAttrib in this example)! So it is not possible to define one handler for e.g. two attributes **visible** and **numDecimalPlaces** because one handler would expect a boolean and one a number (unless you want to cast).
* The child elements (e.g. all fields in a **form**) are also an attribute of the form and therefore handled using an attribute handler as well. In that case the input parameter is a list of the child elements:  
    
  @SamlAttributeHandler(handles = {  
   @SamlAttribute(elementType=FormElement.**class**, featureId=SamlPackage.*FORM\_ELEMENT\_\_FIELDS*)  
  })  
  **public** **void** dispatchFieldElements(List<FormFieldElement> fields) **throws** CoreException {  
   …  
  }

#### SAML Context

The SAML element importers and attribute handlers use a very modular and extensible approach without tight connections. The SAML Context is the component that brings these modules together again.

A SAML Context lives during one import and is accessible everywhere. It contains access to progress monitor, working copies, the SAML grammar, the current element stack and much more. Basically it gives the different importer modules the opportunity to share and exchange data.

#### Entry Points

The basic entry point where an import starts is the class org.eclipse.scout.sdk.saml.importer. SamlImportHelper. It contains methods to start an import sync (blocking) or async, with or without having the Eclipse workbench running. It is used

* by the import wizard (org.eclipse.scout.sdk.saml.importer.ui.wizard.SamlImportWizard) used when manually invoking the import from the Scout Explorer (<http://wiki.eclipse.org/Scout/SDK/Explorer_View>).
* the SAML Import JUnit Plugin Tests
* and the SAML Import Application (org.eclipse.scout.sdk.saml.importer.application. SamlImportApplication) used when headlessly running an import e.g. from Ant (see 4.3.3).

These three entry points make all use of the SamlImportHelper which actually starts the org.eclipse.scout.sdk.saml.importer.operation.SamlImportOperation in which the elements are imported.

### Generated Artifacts

This chapter describes the artifacts that are generated by the Importer based on a specific input. The following table gives an overview over all possibly generated artifacts. Afterwards each artifact is explained in more detail.

| **Nr.** | **Artifact** | **Description** | **Tier** |
| --- | --- | --- | --- |
|  | Client Service Interface | Contains all client logic method descriptions that can be accessed by the service consumer. | Client |
|  | Client Service | Implementation of the corresponding interface defined in G01.01. | Client |
|  | Process Service Interface | Contains all server-logic method descriptions that can be accessed by the service consumer. | Shared |
|  | Process Service | Implementation of the corresponding interface defined in G01.03. | Server |
|  | Code Type | Static code definition to be used e.g. on smartfields. | Shared |
|  | Form | Represents a UI dialog or form containing form field widgets, validation and event-handling (calling client- or server-logic services). | Client |
|  | Form Data | DTO to be used between the UI on the client and the service facade on the server. | Shared |
|  | Lookup Service Interface | Contains all lookup-logic method descriptions that can be used by e.g. a smartfield to fill key-value-pair lists. | Shared |
|  | Lookup Service | Implementation of the corresponding interface defined in G01.09. | Server |
|  | Lookup Call | DTO for a lookup service. Contains all parameters to be used by the corresponding lookup service. | Shared |
|  | Texts\_\*.properties | NLS resources containing the translations for a specific locale. | Shared |
|  | Text Provider Service | Contains the logic to access a set of Texts\_\*.properties files. | Shared |
|  | Preferences | Contains Scout Development Preferences for a Scout Bundle. | All |

#### G01.01 Client Service Interface

|  |  |
| --- | --- |
| Input | **module** ${module}  **form** ${formId} {  **logic** **placement=client** {  "${clientLogicCode}"  }  **logic** **placement=client** **event=init** {  "${clientEventLogicCode}"  }  } |
| Processing | * For each form element in the SAML file a corresponding client service interface is created. * For all event logic elements in the surrounding form with placement=client a method with a matchig signature is created in the client service interface. |
| Output | **package** ${module}.client.services;  **public** **interface** I${formId}ClientService **extends** IService2 {  **public void** ${formId}FormInit(${formId}Form form) **throws** ProcessingException;  } |
| Target Dir | ${module}.client/src/${module}/client/services |
| File name | I${formId}ClientService.java |

#### G01.02 Client Service

|  |  |
| --- | --- |
| Input | **module** ${module}  **form** ${formId} {  **logic** **placement=client** {  "${clientLogicCode}"  }  **logic** **placement=client** **event=init** {  "${clientEventLogicCode}"  }  […]  } |
| Processing | * For each form element in the SAML file a corresponding client service is created. * For all event logic elements in the surrounding form with placement=client a method is added containing the given source. * For all non-event logic elements in the surrounding form the given source code is added to the service on top level. |
| Output | **package** ${module}.client.services;  **public** **class** ${formId}ClientService **extends** AbstractService  **implements** I${formId}ClientService {  "${clientLogicCode}"  @Override  **public void** ${formId}FormInit(${formId}Form form) **throws** ProcessingException {  "${clientEventLogicCode}"  }  } |
| Target Dir | ${module}.client/src/${module}/client/services |
| File name | ${formId}ClientService.java |

#### G01.03 Process Service Interface

|  |  |
| --- | --- |
| Input | **module** ${module}  **form** ${formId} {  **logic** **placement=server** {  "${serverLogicCode}"  }  **logic** **placement=server** **event=modify\_load** {  "${serverEventLogicCode}"  }  […]  } |
| Processing | * For each form element in the SAML file a corresponding process service interface is created. * For all event logic elements in the surrounding form with placement=server a method with a matchig signature is created in the process service interface. |
| Output | **package** ${module}.shared.services.process;  **public** **interface** I${formId}ProcessService **extends** IService2 {  **public** ${formId}FormData load(${formId}FormData formData) **throws** ProcessingException;  } |
| Target Dir | ${module}.shared/src/${module}/shared/services/process |
| File name | I${formId}ProcessService.java |

#### G01.04 Process Service

|  |  |
| --- | --- |
| Input | **module** ${module}  **form** ${formId} {  **logic** **placement=server** {  "${serverLogicCode}"  }  **logic** **placement=server** **event=modify\_load** {  "${serverEventLogicCode}"  }  […]  } |
| Processing | * For each form element in the SAML file a corresponding process service is created. * For all event logic elements in the surrounding form with placement=server a method is added containing the given source. * For all non-event logic elements in the surrounding form the given source code is added to the service on top level. |
| Output | **package** ${module}.server.services.process;  **public** **class** ${formId}ProcessService **extends** AbstractService  **implements** I${formId}ProcessService {  "${serverLogicCode}"  @Override  **public** ${formId}FormData load(${formId}FormData formData) **throws** ProcessingException {  "${serverEventLogicCode}"  }  } |
| Target Dir | ${module}.server/src/${module}/server/services/process |
| File name | ${formId}ProcessService.java |

#### G01.05 Code Type

|  |  |
| --- | --- |
| Input | **module** ${module}  **code** ${codeName} **id**=1112 |
| Processing | * For each code element in the SAML file a corresponding code type class is created. |
| Output | **package** ${module}.shared.services.code;  **public** **class** ${codeName}CodeType **extends** AbstractCodeType<Integer> {  **public** **static** **final** Integer *ID* = 1112;  @Override  **public** Integer getId() {  **return** *ID*;  }  } |
| Target Dir | ${module}.shared/src/${module}/shared/services/code |
| File name | ${codeName}CodeType.java |

#### G01.06 Form

|  |  |
| --- | --- |
| Input | **form** ${formId} {  **sequence\_box** ${boxId} {  **string** ${stringFieldId}  […]  }  […]  } |
| Processing | * For each form element in the SAML file a corresponding form class is created. |
| Output | **package** ${module}.client.ui.forms;  **public** **class** ${formId}Form **extends** AbstractForm {  […]  } |
| Target Dir | ${module}.client/src/${module}/client/ui/forms |
| File name | ${formId}Form.java |

#### G01.07 Form Data

|  |  |
| --- | --- |
| Input | **form** ${formId} {  **sequence\_box** ${boxId} {  **string** ${stringFieldId}  […]  }  […]  } |
| Processing | * For each form element in the SAML file a corresponding form data class is created. |
| Output | **package** ${module}.shared.services.process;  **public** **class** ${formId}FormData **extends** AbstractFormData {  […]  } |
| Target Dir | ${module}.shared/src/${module}/shared/services/process |
| File name | ${formId}FormData.java |

#### G01.08 Lookup Service Interface

|  |  |
| --- | --- |
| Input | **lookup** ${lookupId} {  **logic** **placement**=**server** {  "${lookupServiceLogic}"  }  **logic** **event**=**all** **placement**=**server** {  "${lookupServiceEventLogic}"  }  } |
| Processing | * For each lookup element in the SAML file a corresponding lookup service interface is created. |
| Output | **package** ${module}.shared.services.lookup;  **public** **interface** I${lookupId}LookupService **extends** ILookupService {  } |
| Target Dir | ${module}.shared/src/${module}/shared/services/l |
| File name | I${lookupId}LookupService.java |

#### G01.09 Lookup Service

|  |  |
| --- | --- |
| Input | **lookup** ${lookupId} {  **logic** **placement**=**server** {  "${lookupServiceLogic}"  }  **logic** **event**=**all** **placement**=**server** {  "${lookupServiceEventLogic}"  }  } |
| Processing | * For each lookup element in the SAML file a corresponding lookup service is created. * For all event logic elements in the surrounding lookup with placement=server a method is added containing the given source. * For all non-event logic elements in the surrounding lookup the given source code is added to the service on top level. |
| Output | **package** ${module}.server.services.lookup;  **public** **class** ${lookupId}LookupService **extends** AbstractLookupService  **implements** I${lookupId}LookupService {  "${lookupServiceLogic}"  @Override  **public** LookupRow[] getDataByAll(LookupCall call) **throws** ProcessingException {  "${lookupServiceEventLogic}"  }  } |
| Target Dir | ${module}.server/src/${module}/server/services/lookup |
| File name | ${lookupId}LookupService.java |

#### G01.10 Lookup Call

|  |  |
| --- | --- |
| Input | **lookup** ${lookupId} {  **logic** **placement**=**server** {  "${lookupServiceLogic}"  }  **logic** **event**=**all** **placement**=**server** {  "${lookupServiceEventLogic}"  }  } |
| Processing | * For each lookup element in the SAML file a corresponding lookup call class is created. |
| Output | **package** ${module}.shared.services.lookup;  **public** **class** ${lookupId}LookupCall **extends** LookupCall {  @Override  **protected** Class<? **extends** ILookupService> getConfiguredService() {  **return** I${lookupId}LookupService.**class**;  }  } |
| Target Dir | ${module}.shared/src/${module}/shared/services/lookup |
| File name | ${lookupId}LookupCall.java |

#### G01.11 Texts.properties

|  |  |
| --- | --- |
| Input | **translation** ${translationId} default\_lang="${txt1}" ${langIsoCode2}="${txt2}" […] |
| Processing | * For each distinct language (language iso codes) in all translation elements a corresponding properties file is created * For the special language ‘default\_lang’ a default properties file is created. This translation is used if a translation does not exist for a specific language and must be provided. * For each translation element the given texts are inserted in the corresponding properties files created above. |
| Output | **Texts.properties:**  ${translationId}=${txt1}  **Texts\_${langIsoCode2}.properties:**  ${translationId}=${txt2} |
| Target Dir | ${module}.shared/resources/texts/ |
| File name | Texts\_${langIsoCode1}.properties Texts\_${langIsoCode2}.properties […] |

#### G01.12 Text Provider Service

|  |  |
| --- | --- |
| Input | **module** ${moduleId}  **translation** ${translationId} default\_lang="${txt1}" ${langIsoCode2}="${txt2}" […] |
| Processing | * For each module that contains translation elements a corresponding TextProviderService is created if not yet existing. * The service is configured to point to the corresponding translation resources (see G01.11). |
| Output | **package** ${module}.shared.services.common.text;  **public** **class** ${moduleId}TextProviderService **extends** AbstractDynamicNlsTextProviderService{  @Override  **protected** String getDynamicNlsBaseName() {  **return** "resources.texts.Texts";  }  } |
| Target Dir | ${module}.shared/src/${module}/shared/services/common/text |
| File name | ${moduleId}TextProviderService.java |

#### G01.13 Preferences

|  |  |
| --- | --- |
| Input | No Input required. All Scout Projects in the workspace are automatically affected. |
| Processing | For each Scout Project in the Workspace a Scout SDK Preference file is created.  It contains information about how source code generated with the Scout SDK will look like by default.  This mechanism is used by the SAML Importer to specify e.g. the default super classes that should be used. |
| Output | eclipse.preference.version=1  prefKey1=prefValue1  […] |
| Target Dir | ${bundleName}/.settings/ |
| File name | org.eclipse.scout.sdk.prefs |

### Headless Run

The SAML Import can also be executed without the Eclipse UI (headless mode). This can be done using Windows batch, Unix shell or Ant (<http://ant.apache.org/>). Sample scripts can be found under [4]. The different input parameters are documented in [5].

The run requires an Eclipse 3.8.2 Classic (that matches the operating system where you want to execute the import) with several additional components:

1. Eclipse Scout Runtime 3.9.0 or newer  
   Install from <http://download.eclipse.org/scout/releases/3.9> using P2.
2. Eclipse Scout SDK 3.10.0 or newer  
   Install from <http://download.eclipse.org/scout/nightly> using P2.
3. Eclipse Xtext 2.3.1  
   Install from <http://download.eclipse.org/releases/juno> using P2.
4. SAML 1.0.0, SAML Importer 3.10.0 and optional some customer specific modifications.  
   There is no official SAML & SAML Importer P2 update site yet.

Giving the import process enough memory can drastically improve import performance. Therefore it is highly recommended to run the import with an x64 JRE 7 and therefore an x64 version of Eclipse. Depending on your workspace size and the amount of files to import memory can be between 1 and 5GB.

During the import warning- and error messages are directly shown in the console. More detailed log can be found in the Eclipse workspace log: <your\_import\_workspace>/.metadata/.log

### Known Problems

This chapter contains known problem of the SAML import, explains why this problem occurs and how to solve it (if necessary or possible).

#### Super class of <type> could not be found

|  |  |
| --- | --- |
| Symptom | Exception "Super class of <any\_type> could not be found." occurs in org.eclipse.scout.sdk. operation.method.MethodOverrideOperation.run(). |
| Reason | The bundle classpath is invalid. Either the requested class is not on the classpath of the bundle (missing dependency in the bundle) or there are several types with the same fully qualified name in the workspace. |
| Solution | Close the "Virgo Runtime at …" project in your Eclipse workspace and ensure the manifests of your bundles are correct. |

#### Unable to load secure storage module (<module name>)

|  |  |
| --- | --- |
| Symptom | Exception "org.eclipse.equinox.security.storage.StorageException: Unable to locate secure storage module (<module name>)". |
| Reason | When starting Eclipse it tries to load all encrypted passwords in the secure\_store of eclipse. The store exists by user (by default in C:\Users\<username>\.eclipse\org.eclipse.equinox.security under windows). The passwords are stored in encrypted form and the encryption process uses a native library. Therefore passwords encrypted with an x86 Eclipse cannot be decrypted by a x64 Eclipse and vise verca. |
| Solution | Rename the folder containing the secure\_store so that the Eclipse does not find any encrypted passwords and therefore does not try to decrypt them and rename it back afterwards.  Or just ignore the Exception. |

#### IllegalArgumentException

|  |  |
| --- | --- |
| Symptom | Exception "java.lang.IllegalArgumentException” in org.eclipse.jdt.internal.formatter.DefaultCodeFormatter. |
| Reason | Bug in JDT while formatting the source of a Java class. |
| Solution | None. Just ignore the Exception. |

# List of Abbreviations

| **Abbreviation** | **Explanation** |
| --- | --- |
| AST | Abstract Synthax Tree A model describing a synthax hierarchy to the lowest levels. |
| SAML | Scout Application Modeling Language |
| DSL | Domain specific language |
| EMF | Eclipse Modeling Framework. <http://www.eclipse.org/modeling/emf/> |
| SDK | Software Development Kit Set of software development tools that allows to create applications with a certain technology or language. |
| API | Application Programming Interface Specification intended to be used as an interface by software components to communicate with each other. |
| IDE | A software application that provides comprehensive facilities and tools to computer programmers for software development. |
| DTO | Data transfer object D design pattern used to transfer data between software application subsystems or tiers. |
| Smartfield | A Scout specific, combo box like UI widget that includes search-as-you-type behavior. |
| UI | User Interface Graphical interface between a human user and the machine taking the commands. |
| OS | Operating System A software that manages computer hardware resources and provides common services for computer programs. |

# Referenced Documents

| **Document** | **Description** |
| --- | --- |
| 1. Sample Files | <https://github.com/BSI-Business-Systems-Integration-AG/org.eclipse.scout.xtext/tree/develop/org.eclipse.scout.saml/saml_samples> |
| 1. SAML definition as Xtext | <https://github.com/BSI-Business-Systems-Integration-AG/org.eclipse.scout.xtext/blob/develop/org.eclipse.scout.saml/src/org/eclipse/scout/saml/Saml.xtext> |
| 1. SAML Release Notes | <https://github.com/BSI-Business-Systems-Integration-AG/org.eclipse.scout.xtext/blob/develop/org.eclipse.scout.saml/release_notes.txt> |
| 1. SAML Headless Script Samples | <https://github.com/BSI-Business-Systems-Integration-AG/org.eclipse.scout.xtext/tree/develop/org.eclipse.scout.sdk.saml.importer/headless> |
| 1. SAML Headless Ant Macro | <https://github.com/BSI-Business-Systems-Integration-AG/org.eclipse.scout.xtext/blob/develop/org.eclipse.scout.sdk.saml.importer/headless/ant/macros/saml.xml> |