Dokumentation Projektpraktikum

T3editor – Codecompletion

Christian Kartnig

Matr.-Nr.: 9671147

St.-Kz.: 033 534

SS 2008

Inhaltsverzeichnis

Einleitung

Dieses Dokument beschreibt die Entwicklung einer Codecompletion-Software für TypoScript und dokumentiert die Software.

TypoScript ist die Konfigurationssprache des bekannten Open Source CMS TYPO3.
Die Struktur einer TYPO3-Website wird in TYPO3 durch den Seitenbaum gebildet, einer baumförmigen Hierarchie, deren Knoten die einzelnen Seiten-Datensätze der Datenbank repräsentieren.

Jeder dieser Knoten kann mit einem sogenannten TypoScript Template verknüpft werden, einem Datensatz, der TypoScript-Anweisungen beinhaltet, die spezifizieren, wie die betreffende Seite gerendert werden soll. Der TypoScript-Code wird automatisch im Seitenbaum bis in die Blattknoten des jeweiligen Zweiges nach unten vererbt, kann in Unterseiten aber überschrieben werden.

Die Tatsache, dass TypoScript-Code über die ganze Website verteilt geschrieben wird und gezielt vererbt wird, ist meiner Ansicht nach einer der Gründe, warum TypoScript hauptsächlich direkt über den Browser am Server editiert wird und gleichzeitig auch eine der Ursachen der Probleme, die Entwickler mit TypoScript haben. Durch die Verteilung des Codes verliert man schnell den Überblick, welche Eigenschaften schon gesetzt wurden - der Wechsel in den Object Browser unterbricht durch Seitenreloads den Arbeitsfluss.

Deshalb wollten Stephan Petzl und ich eine kontextabhängige Codecompletion für TypoScript entwickeln, die im Browser läuft und in das TYPO3-Backend integrierbar ist.

Wir beschlossen, die Codecompletion auf den sich damals gerade in Entwicklung befindenden TypoScript Editor aufzusetzen, eine System Extension, die seit der Version 4.2 Bestandteil der TYPO3-Distribution ist. Vielen Dank an dieser Stelle an Tobias Liebig, den Entwickler des TypoScript Editors, für die gute Kooperation. Außerdem möchten wir Bernhard Kraft danken, von dem wir eine erste Version der XML-TypoScript-Referenz übernahmen.

Anforderungen

Funktionale Anforderungen

Es soll eine Codecompletion programmiert werden, die im Browser das Schreiben von TypoScript Code unterstützt. Beim Schreiben des Zeichens ".", also des Punkts, soll ein Fenster erscheinen, das möglichst sinnvolle Vorschläge enthält. Außerdem soll die Codecompletion explizit durch Ctrl-Space getriggert werden können.

Die Vorschläge der Codecompletion sollen per Maus oder Tastatur ausgewählt werden können und sollen dann automatisch an der Cursorposition in den Text eingefügt werden.

Die Vorschläge sollen vordefinierte TypoScript-Typen und Eigenschaften sowie benutzerdefinierte (d.h. durch ihre Verwendung definierte) Eigenschaften beinhalten.

Die Referenz sowohl der vordefinerten TypoScript-Typen und Eigenschaften, als auch von Extensions eingeführter Typen und Eigenschaften sollen einfach durch eine XML-Datei eingebunden werden.

Es soll bei der Berechnung der Vorschläge der Code der momentanen Version des aktuell geöffneten TypoScript Templates von Anfang des Datensatzes bis zur Cursorposition einbezogen werden. Code unterhalb der Cursorposition soll ignoriert werden, um den Benutzer nicht zu verwirren. Außerdem sollen alle TypoScript Templates, die in der Seitenbaum-Hierarchie über dem aktuellen Template stehen, mit einbezogen werden.

Es sollen ausschliesslich TypoScript-Pfade vervollständigt werden. Vorschläge für Operatoren oder Anzeige erlaubter Wertzuweisungen könnten später als Zusatzfeatures implementiert werden, sollen aber in dieser Version der Software keine Beachtung finden.

Kommentare sollen ignoriert werden. TypoScript-Bedingungen (conditions) sind zur Entwicklungszeit im Allgemeinen unentscheidbar. Deswegen soll TypoScript-Code innerhalb von Conditions ignoriert werden, da nicht davon ausgegangen werden kann, dass der Code überhaupt eingebunden wird. Mehrzeilige Wertzuweisungen sollen ebenfalls ignoriert werden, da Werte im Codebaum nur gespeichert werden, um eventuelle Eigenschaften vordefinierter Typen auflösen zu können und mehrzeilige Werte keine Typzuweisung beinhalten.

Nichtfunktionale Anforderungen

- Technologie/Sprachwahl
  Durch die Entscheidung, die Codecompletion in den t3editor von Tobias Liebig zu integrieren, ist auch die Entscheidung für die verwendeten Technologien und Sprachen gefallen: PHP und die TYPO3 API für Extensions auf der Serverseite, Javascript und das Prototype-Framework (das auch der t3editor verwendet) auf der Client-Seite.

- Performance
  Das Laden der Daten und der TypoScript-Referenz vom Server, sowie das Parsen des Codes dürfen die Benutzung des Editors nicht durch Verzögerungen behindern. Das Triggern der Codecompletion darf auch bei viel Code keine merkbaren Wartezeiten produzieren.

- Lose Koppelung
  Die Implementierung der Codecompletion muss eine lose Koppelung zu anderen beteiligten Komponenten besitzen. Die Integration in TYPO3 und den t3editor soll mit klaren, möglichst minimalen Schnittstellen realisiert werden. Änderungen der anderen Komponenten sollen möglichst wenig Einfluss auf die Codecompletion haben, ebenso sollen Änderungen in der Codecompletion die anderen Komponenten möglichst unbeeinflusst lassen.

- Erweiterbarkeit
  Zusätzliche Funktionalität soll leicht in den bestehenden Code integrierbar sein, auch von anderen Entwicklern sollen leicht Zusatz-Features implementiert werden können.

Implementierung

Die Codecompletion wurde in den t3editor von Tobias Liebig integriert, der sich zum Zeitpunkt des Beginns der Implementierung gerade in Entwicklung befand. Der t3editor basiert auf einem für diesen Zweck angepassten Branch des Codemirror-Editors, ein von Marijn Haverbeke entwickelter Javascript-Editor [http://marijn.haverbeke.nl/codemirror/]. Codemirror stellt bereits Funktionen für Einrückung und Formatierung und Schnittstellen für Syntax-Highlighting zur Verfügung.

Wir entwickelten die Codecompletion auf Basis einer Betaversion des t3editors. Die Version, die nun in der Distribution der Version 4.2 von TYPO3 zu finden ist, weicht an etlichen Stellen fundamental von der von uns verwendeten Betaversion ab, was uns viel Entwicklungszeit kostete, da Features der Codecompletion mit der aktuellen Version plötzlich nicht mehr funktionierten und angepasst bzw. zum Teil neu programmiert werden mussten.

Um eine lose Koppelung zu erreichen, wurde die Kommunikation zwischen Editor und Codecompletion auf ein Minimum begrenzt, nämlich die Instanzierung der zentralen tsCodeCompletion-Klasse und der Registrierung dreier Eventhandler für die Behandlung von Tastendruck und Mausklicks. Um nach der Instanzierung ein sofortiges Editieren zu erlauben, werden die für die Codecompletion benötigten Daten asynchron per AJAX nachgeladen.

Wir implementierten einen Plugin-Mechanismus, der es anderen Entwicklern ermöglicht, ihre Erweiterungen als herkömmliche TYPO3-Extensions einzubinden. Es werden Hooks an wichtigen Stellen der Codecompletion und ein Plugin-Context mit nützlichen Teilen des Environments zur Verfügung gestellt. Gewissermaßen exemplarisch wurde die Anzeige von Beschreibungen zu den TypoScript-Eigenschaften als Plugin realisiert. Dieses Plugin wird allerdings nicht per Hook aus einer externen Extension eingebunden, sondern direkt im Code instanziert, damit nicht die Installation einer zusätzlichen Extension für diese Funktionalität notwendig wird.

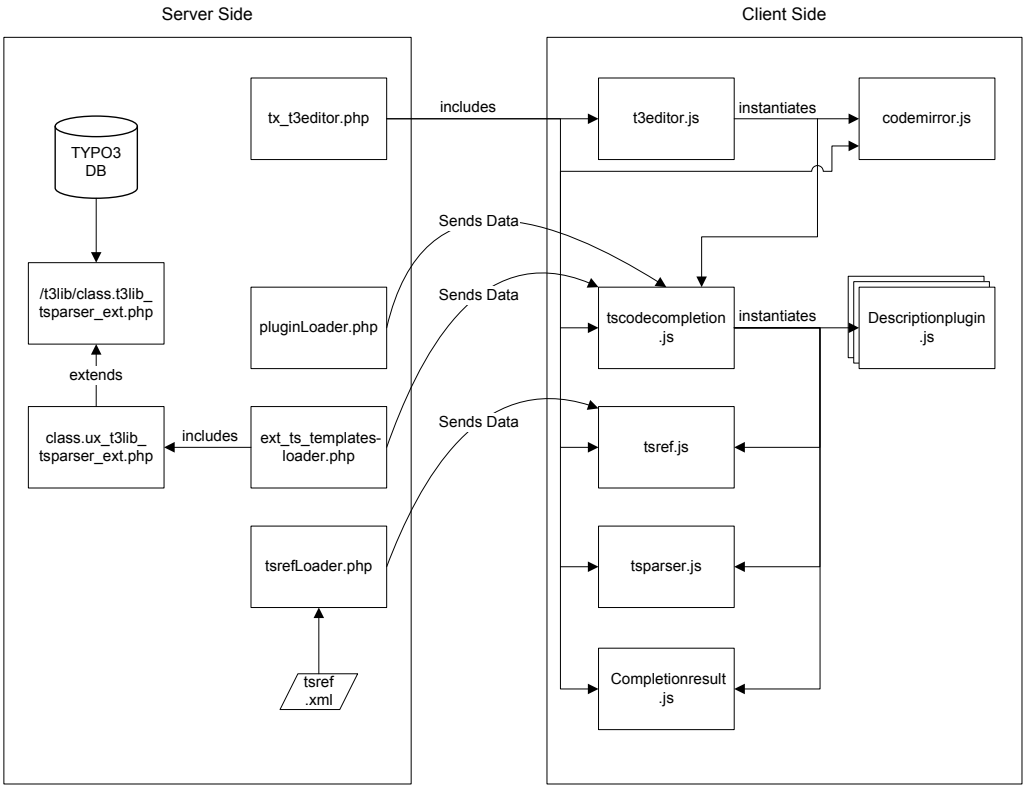
Struktur

Die notwendigen Javascript-Dateien zum Betrieb der Codecompletion werden in der PHP-Datei der t3editor-Extension eingebunden (tx_t3editor.php). In t3editor.js wird ein Objekt der Klasse tscodecompletion instanziiert, die ihrerseits alle restlichen Objekte der Codecompletion instanziiert.

Die tscodecompletion-Klasse kommuniziert mit dem serverseitigen Script `pluginLoader.php` und bekommt von diesem die Namen und Dateipfade von allfälligen registrierten Plugins übermittelt (Entwickler haben in ihren Extensions die Möglichkeit, im globalen Array `$TYPO3_CONF_VARS['EXTCONF']['t3editor']['plugins']` den Ort und Namen ihres Plugins zu registrieren, `pluginLoader.php` liest dieses Array aus). Anschliessend wird jedes Plugin von der tscodecompletion-Klasse eingebunden und instanziiert.

Der Code der externen Templates (der Code der TypoScript Templates, die in der Seitenbaum-Hierarchie über dem aktuellen Template stehen) wird ebenfalls asynchron per HTTPRequest nachgeladen. Das Auslesen und Übermitteln an die tscodecompletion-Klasse übernimmt das PHP-Script `ext_ts_templatesloader.php`. Zum Auslesen verwendet das Script Funktionen aus der t3lib-Bibliothek, nämlich aus `class.t3lib_tsparser_ext.php`. Allerdings liest die verwendete Funktion sämtliche Templates in der Rootline ein, also auch das aktuell bearbeitete. Das ist für die Codecompletion nicht brauchbar, es soll ja vom aktuellen Template nur der Code, der gerade im Editor bearbeitet wird, bis zur aktuellen Cursorposition berücksichtigt werden, und nicht eine bereits abgespeicherte Vorgängerversion. Deswegen wird die Klasse `ux_class.t3lib_tsparser_ext.php` verwendet, die von der Originalklasse erbt, aber das aktuelle Template nicht ausliest.

Die tsref-Klasse bekommt vom PHP-Script `tsrefLoader.php` die TypoScript-Referenz XML-Datei im JSON-Format übermittelt.



Komponenten

tscodecompletion

Diese Klasse ist das Kernstück der Codecompletion. Sie wird direkt vom Editor instanziiert. Sie instanziiert alle übrigen Klassen, steuert alle Abläufe und übernimmt auch die Darstellung der Codecompletion-Box. Dem Konstruktor werden zwei Parameter übergeben, eine Instanz des codeMirror-Objektes, die im Moment hauptsächlich zur Ermittlung der aktuellen Cursorposition gebraucht wird, und outerdiv, das HTML-Element, das den Editor beinhaltet, das zur DOM-Manipulation gebraucht wird. Der Konstruktor lädt die TypoScript-Referenz, die externen Templates und eventuelle Plugins (siehe oben).

TypoScript-Referenzen werden in TYPO3 nicht aufgelöst, es wird nur ein String gespeichert, auf welchen Knoten die Referenz verweist. Referenzen in externen Templates werden deswegen von der Methode `resolveExtReferencesRec()` aufgelöst.

Wird ein Punkt geschrieben oder Ctrl-Space gedrückt, so rufen die in `t3editor.js` registrierten Eventhandler die Methode `refreshCodeCompletion()` auf, die den Parser anweist, den Baum neu aufzubauen und ein neues Objekt der Klasse `completionresult` erzeugt. Die in `completionresult` gefundenen Vorschläge werden dann von der Methode dargestellt.

tsparser

Die Parserklasse bekommt im Konstruktor die TypoScript-Referenz und den Codebaum der externen Templates übergeben. Ihre einzige öffentliche Methode `buildTsObjTree()` erwartet als Argumente die DOM Node des ersten Tokens im Editor und die DOM Node des Tokens an der Cursorposition. Sequentiell werden die einzelnen Knoten durchgegangen und ein Codebaum wird aufgebaut. Dazu wird die Klasse `TreeNode` verwendet, die ebenfalls in `tsparser.js` definiert ist. Eine TreeNode speichert ihren Namen, ihren Typ bzw. Wert, sowie Referenzen auf ihren Elternknoten und alle Kindknoten. Die TreeNode an der Cursorposition ist der Rückgabewert der Methode `buildTsObjTree()`.

TsRef

Diese Klasse ist verantwortlich für das asynchrone Nachladen der TypoScript Referenz. Diese ist am Server in einer XML-Datei abgelegt, die folgende Struktur hat:

```
<?xml version="1.0"?>
<tsref>
 <TYPE1 name="TYPE1" extends="otherType">
  <property name="prop1" type="typeX" />
  <property name="prop2" type="typeY" />
 </TYPE1>

 <TYPE2 name="TYPE2">
  <property name="prop3" type="typeZ" />
  .
  .
  .
</tsref>
```

Das Attribut `extends` verweist auf einen anderen Typ, dessen Eigenschaften übernommen werden.

Die Klasse `TsRef` lädt mit Hilfe des serverseitigen Scripts `tsrefLoader.php` die TypoScript-Referenz im JSON-Format nach und baut einen Baum auf. Dazu werden die beiden Containerklassen `TsRefType` und `TsRefProperty` verwendet, die die einzelnen Knoten im Baum repräsentieren.

Weiters stellt die Klasse Methoden für den Zugriff auf den Baum zur Verfügung.

CompletionResult

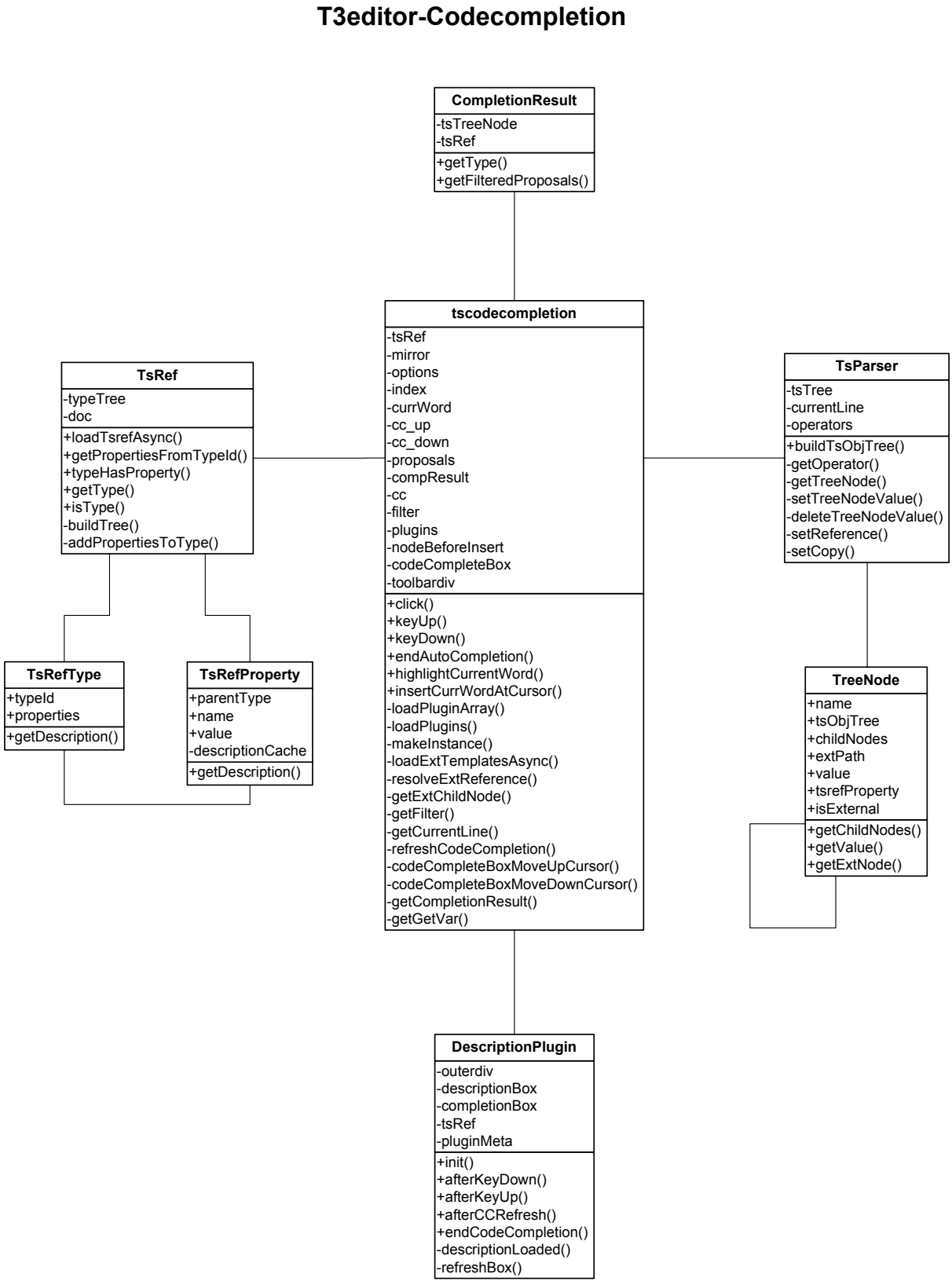
Die Klasse `CompletionResult` führt die Ergebnisse der Codecompletion zusammen und kapselt die ermittelten Vorschläge. Sie bekommt den zuvor im Parser ermittelten aktuellen Knoten im Codebaum und die TsRef übergeben. Sie stellt zwei öffentliche Methoden zur Verfügung. `getTpe()` liefert den Typ des aktuellen Knotens zurück und `getFilteredProposals()` gibt die vorhandenen Vorschläge zurück. Die Methode befüllt ein Array mit den Kindknoten des aktuellen Knotens und den Eigenschaften, die in der TsRef für den Typ des aktuellen Knotens eingetragen sind. Die Objekte im Array haben die Eigenschaften `word`, `cssClass` und `type`.

`getFilteredProposals()` erwartet als Argument den String `filter`. Die Methode gibt nur die Eigenschaften zurück, deren Namen mit dieser Zeichenkette beginnt. Damit wird ein Caching-Mechanismus beim “Type-On” realisiert. Wenn der Benutzer nach dem Triggern der Codecompletion weitertippt, muss nicht bei jedem Tastendruck der Codebaum neu generiert werden, sondern es werden einfach die Einträge gefiltert. Beim Verlassen der Node (z.B. durch Tippen eines weiteren Punktes oder Löschen des vorangegangenen) muss der Codebaum natürlich neu berechnet werden.

DescriptionPlugin

Das DescriptionPlugin ist mit dem Plugin-Mechanismus der Codecompletion realisiert. Es implementiert die vier Interceptor-Methoden `init()`, `afterKeyDown()`, `afterKeyUp()` und `afterCCRefresh()` und nutzt den übergebenen Plugin-Context, der aus dem Parser, der Codecompletion-Box, dem Eltern-DOM-Element des Editors, der TsRef sowie Referenzen auf andere Plugins besteht. Es zeigt neben der Codecompletion-Box eine Box mit Beschreibungen zur jeweiligen Eigenschaft an.

Klassendiagramm



Testfälle

Es wurde manuelles Blackbox Testing auf die Software durchgeführt. Anschliessend sind die Testergebnisse abgebildet. Die in grün markierten Punkte entsprechen der Eingabe die durch den Tester gemacht werden müssen.

Titel	Eingabe	Ausgabe	Test bestanden?
(1) TSREF Attribute Es wird überprüft ob die TypoScript Referenz geladen, und der Typ korrekt ermittelt wurde	temp.myImage = IMAGE temp.myImage.	Alle TypoScript Eigenschaften des Datentyps "IMAGE" (siehe TypoScript Referenz []) Die Eigenschaften müssen orange markiert sein.	ja
(2) Benutzerdefinierte Attribute Es wird überprüft ob die benutzerdefinierten Attribute korrekt geparsed und angezeigt werden	temp.myImage = IMAGE temp.myImage.myProperty = 1 temp.myImage.	Alle TypoScript Eigenschaften des Datentyps "IMAGE", und die benutzerdefinierte Eigenschaft "myProperty". Diese muss grau markiert sein.	ja

(3) Klammer auflösung
Es wird getestet ob Klammern beim Parsen richtig interpretiert werden
	temp.myImage = IMAGE
	temp{
	my.
	}
		Alle TypoScript Eigenschaften des Datentyps "IMAGE". Die Eigenschaften müssen orange markiert sein.	ja

(4.1) Kopieren 1 Es wird überprüft ob Objektbäume korrekt kopiert werden	temp.myImage = IMAGE temp.myImage2 < temp.myImage temp.myImage2.xxx = 1 temp.myImage2.	Alle TypoScript Eigenschaften des Datentyps "IMAGE", und die Benutzerdefinierte Eigenschaft "xxx".	ja
(4.2) Kopieren 2 Es wird überprüft ob Kopierquelle unberührt bleibt	temp.myImage = IMAGE temp.myImage2 < temp.myImage temp.myImage2.xxx = 1 temp.myImage.	Alle TypoScript Eigenschaften des Datentyps "IMAGE", jedoch ohne die Benutzerdefinierte Eigenschaft "xxx".	ja
(5.1) Referenzieren 1 Es wird überprüft ob Objektbäume korrekt referenziert werden	temp.myImage = IMAGE temp.myImage2 =< temp.myImage temp.myImage2.xxx = 1 temp.myImage2.	Alle TypoScript Eigenschaften des Datentyps "IMAGE", und die Benutzerdefinierte Eigenschaft "xxx".	ja
(5.2) Referenzieren 1 Es wird überprüft ob sich Änderungen am Referenzierungsziel korrekt auf die Referenzquelle auswirken	temp.myImage = IMAGE temp.myImage2 =< temp.myImage temp.myImage2.xxx = 1 temp.myImage.	Alle TypoScript Eigenschaften des Datentyps "IMAGE", und die Benutzerdefinierte Eigenschaft "xxx".	ja
(6.1) Löschen 1 Es wird überprüft ob Objekte und Eigenschaften vollständig gelöscht werden und wieder neu definiert werden können	temp.myImage = IMAGE temp.myImage > temp.myImage.test = 4 temp.myImage.	Es darf lediglich die Benutzereigenschaft "test" angezeigt werden.	ja
(6.2) Löschen 2 Es wird überprüft ob Objekbäume vollständig gelöscht werden	temp.myImage = IMAGE temp.myImage.test1.test2 = 10 temp.myImage > temp.	Es darf nichts angezeigt werden, die Codecompletion Anzeige bleibt unsichtbar.	ja

(7) relative Pfade
Es wird überprüft ob relative Pfade richtig aufgelöst werden
	temp.myImage = IMAGE
	temp.myImage{
	[strg+space]
	}
		Alle TypoScript Eigenschaften des Datentyps "IMAGE".	ja

(8) externe Templates Es wird überprüft ob externe Templates geladen wurden	plugin.tx_indexedsearch.	Alle Eigenschaften des Plugins "tx_indexedsearch"	ja
(9) Strg+Leertaste Es wird überprüft ob die Codecompletion mit Strg+Leertaste ausgelöst werden kann	plugin.tx_in[strg+space]	Alle Plugins die mit "tx_in" beginnen	ja
(10) fortlaufendes Tippen Es wird überprüft ob die Ergebnisse korrekt gefiltert werden	plugin.tx_ind	Vorerst alle Plugins, beim Tippen sollte sich die Auswahl entsprechend verringern	ja

JSDoc

t3editor codecompletion

This document is the API Specification for t3editor codecompletion.

Summary

No summary generated for these documents.

File Summary	
completionresult.js	contains the CompletionResult class
descriptionPlugin.js	
tscodecompletion.js	contains the TsCodeCompletion class
tsparser.js	contains the TsParser class and the TreeNode helper class
tsref.js	contains the TsRef class and the TsRefProperty and TsRefType helper classes

Overview File Class **Tree** **Index** **Help** *t3editor codecompletion*
PREV NEXT FRAMES NO FRAMES All Classes

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008

Class CompletionResult

```
Object
   |
   +--CompletionResult
```

class **CompletionResult**

this class post-processes the result from the codecompletion, so that it can be displayed in the next step.
Defined in completionresult.js

Constructor Summary

`CompletionResult(tsRef,tsTreeNode)`	

Method Summary

Object	`getFilteredProposals(<String> filter)` returns a list of possible path completions (proposals), which is: a list of the children of the current TsTreeNode (= userdefined properties) and a list of properties allowed for the current object in the TsRef remove all words from list that don't start with the string in filter
Object	`getType()` returns the type of the currentTsTreeNode

Constructor Detail

CompletionResult

`CompletionResult(tsRef,tsTreeNode)`

Parameters:
`tsRef` - the TsRef Tree
`tsTreeNode` - the current Node in the codetree built by the parser

Returns:
a new CompletionResult instance

Method Detail

getFilteredProposals

```
Object getFilteredProposals(<String> filter)
```

returns a list of possible path completions (proposals), which is: a list of the children of the current TsTreeNode (= userdefined properties) and a list of properties allowed for the current object in the TsRef remove all words from list that don't start with the string in filter

Parameters:

`filter` - beginning of the words contained in the proposal list

Returns:

an Array of Proposals

---

getType

```
Object getType()
```

returns the type of the currentTsTreeNode

Class DescriptionPlugin

```
Object
   |
   +--DescriptionPlugin
```

class **DescriptionPlugin**

Descriptionbox plugin for the t3editor-codecompletion which displays the datatype and the desciption for each property displayed in the completionbox
Defined in descriptionPlugin.js

Constructor Summary

DescriptionPlugin()

Method Summary

void	**afterCCRefresh**(currWordObj,compResult)
void	**afterKeyDown**(currWordObj,compResult)
void	**afterKeyUp**(currWordObj,compResult)
void	**endCodeCompletion**()
void	**init**(pluginContext,plugin)

Constructor Detail

DescriptionPlugin

DescriptionPlugin()

Returns:
A new DescriptionPlugin instance

Method Detail

afterCCRefresh

```
void afterCCRefresh(currWordObj,compResult)
```

---

afterKeyDown

```
void afterKeyDown(currWordObj,compResult)
```

---

afterKeyUp

```
void afterKeyUp(currWordObj,compResult)
```

---

endCodeCompletion

```
void endCodeCompletion()
```

---

init

```
void init(pluginContext,plugin)
```

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008

Class TreeNode

```
Object
   |
   +--TreeNode
```

class **TreeNode**

data structure for the nodes of the code tree mainly used for retrieving the externals templates childnodes
Defined in tsparser.js

Field Summary

Object	**childNodes**
Object	**extPath**
Object	**isExternal**
Object	**name**
Object	**tsrefProperty**
Object	**value**

Constructor Summary

TreeNode(nodeName)

Method Summary

Array	**getChildNodes**() returns local properties and the properties of the external templates
Object	**getExtNode**()
Object	**getValue**()

Field Detail

childNodes

`Object` **`childNodes`**

---

extPath

`Object` **`extPath`**

---

isExternal

`Object` **`isExternal`**

---

name

`Object` **`name`**

---

tsrefProperty

`Object` **`tsrefProperty`**

---

value

`Object` **`value`**

Constructor Detail

TreeNode

`TreeNode``(nodeName)`

Parameters:

name -

Method Detail

getChildNodes

`Array` **`getChildNodes`**`()`

returns local properties and the properties of the external templates

Returns:

ChildNodes

---

getExtNode

```
Object getExtNode()
```

---

getValue

```
Object getValue()
```

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008

Class TsCodeCompletion

```
Object
   |
   +--TsCodeCompletion
```

class **TsCodeCompletion**

This is the main class of the codeCompletion. it is directly invoked by the editor. It instantiates all other classes manages the control flow and takes care of the completionbox
Defined in tscodecompletion.js

Constructor Summary

`TsCodeCompletion(codeMirror,outerdiv)`	Construct a new TsCodeCompletion object.

Method Summary

void	`click(event)` Eventhandler class for mouseclicks ends the codecompletion
void	`endAutoCompletion()` hides codecomplete box and resets completionResult afterwards the interceptor method endCodeCompletion gets called
void	`highlightCurrWord(<int> id)` highlights entry in codecomplete box by id
void	`insertCurrWordAtCursor()` Insert the currently selected item in the proposal list of the codecompletion box into the editor div at cursor position
void	`keyDown(event)` Eventhandler function executed after keystroke release triggers CC on pressed dot and typing on
void	`keyUp(event)` Eventhandler function executed after keystroke release triggers CC on pressed dot and typing on

Constructor Detail

TsCodeCompletion

`TsCodeCompletion``(codeMirror,outerdiv)`

Construct a new TsCodeCompletion object.

Parameters:

`codeMirror` - codeMirror instance, for retrieving the cursor position
`outerdiv` - div that contains the editor, for DOM manipulation

Returns:

A new TsCodeCompletion instance

Method Detail

click

```
void click(event)
```

Eventhandler class for mouseclicks ends the codecompletion

Parameters:

`event` - fired prototype event object

---

endAutoCompletion

```
void endAutoCompletion()
```

hides codecomplete box and resets completionResult afterwards the interceptor method endCodeCompletion gets called

---

highlightCurrWord

```
void highlightCurrWord(<int> id)
```

highlights entry in codecomplete box by id

Parameters:

`id` -

---

insertCurrWordAtCursor

```
void insertCurrWordAtCursor()
```

Insert the currently selected item in the proposal list of the codecompletion box into the editor div at cursor position

See:

- highlightCurrWord()

---

keyDown

```
void keyDown(event)
```

Eventhandler function executed after keystroke release triggers CC on pressed dot and typing on

Parameters:

`event` - fired prototype event object

---

keyUp

```
void keyUp(event)
```

Eventhandler function executed after keystroke release triggers CC on pressed dot and typing on

Parameters:

`event` - fired prototype event object

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008

Class TsParser

```
Object
   |
   +--TsParser
```

class **TsParser**

This class takes care of the parsing and builds the codeTree
Defined in tsparser.js

Constructor Summary

`TsParser(tsRef,extTsObjTree)`
Construct a new TsParser object.

Constructor Detail

TsParser

`TsParser``(tsRef,extTsObjTree)`

Construct a new TsParser object.

Parameters:

`tsRef` - typoscript reference tree
`extTsObjTree` - codeTree for all typoscript templates excluding the current one.

Returns:

A new TsParser instance

Class TsRef

```
Object
   |
   +--TsRef
```

class **TsRef**

This class receives the TsRef from the server and represents it as a tree also supplies methods for access to treeNodes
Defined in tsref.js

Constructor Summary

TsRef()	Construct a new TsRef object.

Method Summary

Object	**getPropertiesFromTypeId**(tId)
Object	**getType**(typeId)
Object	**isType**(typeId)
void	**loadTsrefAsync**()
Object	**typeHasProperty**(typeId,propertyName)

Constructor Detail

TsRef

TsRef()

Construct a new TsRef object.

Returns:
A new TsRef instance

Method Detail

getPropertiesFromTypeId

```
Object getPropertiesFromTypeId(tId)
```

---

getType

```
Object getType(typeId)
```

---

isType

```
Object isType(typeId)
```

---

loadTsrefAsync

```
void loadTsrefAsync()
```

---

typeHasProperty

```
Object typeHasProperty(typeId,propertyName)
```

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008

Class TsRefProperty

```
Object
   |
   +--TsRefProperty
```

class **TsRefProperty**

Represents a TsRefProperty in the tree
Defined in tsref.js

Field Summary

Object	**name**
Object	**parentType**
Object	**value**

Constructor Summary

TsRefProperty(parentType,name,value)

Method Summary

void	**getDescription**(callBack)

Field Detail

name

`Object` **name**

parentType

`Object` **parentType**

value

`Object` **`value`**

Constructor Detail

TsRefProperty

`TsRefProperty``(parentType,name,value)`

Method Detail

getDescription

`void` **`getDescription`**`(callBack)`

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008

Class TsRefType

```
Object
   |
   +--TsRefType
```

class **TsRefType**

Represents a TsRefType in the tree
Defined in tsref.js

Field Summary

Object	**properties**
Object	**typeId**

Constructor Summary

TsRefType(typeId)

Method Summary

void	**getDescription**()

Field Detail

properties

```
Object properties
```

typeId

```
Object typeId
```

Constructor Detail

TsRefType

TsRefType(typeId)

Method Detail

getDescription

void **getDescription**()

Documentation generated by JSDoc on Tue Jul 1 19:52:57 2008