Below is section 11.1.5 as it currently exists in the ES6 draft. Note that the various semantic definitions just run on through the Semantics section although they are roughly grouped according to the production they apply to.

11.1.5 Object Initialiser

NOTE An object initialiser is an expression describing the initialisation of an Object, written in a form resembling a literal. It is a list of zero or more pairs of property names and associated values, enclosed in curly braces. The values need not be literals; they are evaluated each time the object initialiser is evaluated.

Syntax

ObjectLiteral :
  { }  
  { PropertyNameAndValueList }  
  { PropertyNameAndValueList , }

PropertyNameAndValueList :
  PropertyAssignment
  PropertyNameAndValueList , PropertyAssignment

PropertyAssignment :
  IdentifierName
  PropertyName : AssignmentExpression
  PropertyName ( FormalParameterListopt ) { FunctionBody }
  get PropertyName ( ) { FunctionBody }
  set PropertyName ( PropertySetParameterList ) { FunctionBody }

PropertyName :
  IdentifierName
  StringLiteral
  NumericLiteral

PropertySetParameterList :
  BindingIdentifier
  BindingPattern

Semantics

The production ObjectLiteral : { } is evaluated as follows:

1. Return a new object created as if by the expression new Object() where Object is the standard built-in constructor with that name.

The productions ObjectLiteral : { PropertyNameAndValueList } and ObjectLiteral : { PropertyNameAndValueList , } are evaluated as follows:

1. Return the result of evaluating PropertyNameAndValueList.

The PropertyDefinitionList(name) of the production PropertyAssignment : PropertyName : AssignmentExpression is determined as follows:

1. If PropName of PropertyAssignment is not name return the empty List.
2. Return a List containing PropertyAssignment.

The production PropertyNameAndValueList : PropertyAssignment is evaluated as follows:
1. Let \( \text{obj} \) be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
2. Let \( \text{propId} \) be the result of evaluating `PropertyAssignment`.
3. Call the `[[DefineOwnProperty]]` internal method of \( \text{obj} \) with arguments `propId.name`, `propId.descriptor`, and `false`.
4. Return `obj`.

The `PropertyDefinitionList(name)` of the production `PropertyNameAndValueList ; PropertyNameAndValueList , PropertyAssignment` is determined as follows:

1. Let `previous` be `PropertyDefinitionList(name)` of `PropertyNameAndValueList`.
2. If `PropName` of `PropertyAssignment` is `name` then,
   a. Append `PropertyAssignment` to the end of `previous`.
3. Return `previous`.

The static semantics of the production `PropertyNameAndValueList ; PropertyNameAndValueList , PropertyAssignment` are:

- It is a Syntax Error if this production is contained in strict code, `PropertyAssignment` is the production `PropertyAssignment : PropertyName : AssignmentExpression`, and `PropertyDefinitionList(PropName of PropertyAssignment)` of `PropertyNameAndValueList` is not the empty List.
- It is a Syntax Error if `PropertyAssignment` is the production `PropertyAssignment : get PropertyName ( ) { FunctionBody }` and `PropertyDefinitionList(PropName of PropertyAssignment)` of `PropertyNameAndValueList` includes a production of the form `PropertyAssignment : PropertyName : AssignmentExpression`.
- It is a Syntax Error if `PropertyAssignment` is the production `PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }` and `PropertyDefinitionList(PropName of PropertyAssignment)` of `PropertyNameAndValueList` includes a production of the form `PropertyAssignment : PropertyName : AssignmentExpression`.
- It is a Syntax Error if `PropertyAssignment` is the production `PropertyAssignment : get PropertyName ( ) { PropertySetParameterList } { FunctionBody }` and `PropertyDefinitionList(PropName of PropertyAssignment)` of `PropertyNameAndValueList` includes a production of the form `PropertyAssignment : PropertyName : AssignmentExpression`.
- It is a Syntax Error if `PropertyAssignment` is the production `PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }` and `PropertyDefinitionList(PropName of PropertyAssignment)` of `PropertyNameAndValueList` includes a production of the form `PropertyAssignment : PropertyName : AssignmentExpression`.

The production `PropertyNameAndValueList ; PropertyNameAndValueList , PropertyAssignment` is evaluated as follows:

1. Let \( \text{obj} \) be the result of evaluating `PropertyNameAndValueList`.
2. Let \( \text{propId} \) be the result of evaluating `PropertyAssignment`.
3. Call the `[[DefineOwnProperty]]` internal method of \( \text{obj} \) with arguments `propId.name`, `propId.descriptor`, and `false`.
4. Return `obj`.

The `PropName` of the production `PropertyAssignment : IdentifierName` is determined as follows:

1. Return `PropName(IdentifierName)`.

The production `PropertyAssignment : IdentifierName` is evaluated as follows:

1. Let `propName` be `PropName(IdentifierName)`.
2. Let `exprValue` be the result of performing Identifier Resolution as specified in 10.3.1 using `IdentifierName`.
3. Let `propValue` be `GetValue(exprValue)`.

© Ecma International 2009 – All rights reserved
4. Let desc be the Property Descriptor {
   [[Value]]: propValue, 
   [[Writable]]: true, 
   [[Enumerable]]: true, 
   [[Configurable]]: true
}
5. Return Property Identifier (propName, desc).

The PropName of the production `PropertyAssignment : PropertyName : AssignmentExpression` is determined as follows:

1. Return PropName(PropertyName).

The production `PropertyAssignment : PropertyName : AssignmentExpression` is evaluated as follows:

1. Let propName be PropName(PropertyName).
2. Let exprValue be the result of evaluating AssignmentExpression.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor {
   [[Value]]: propValue, 
   [[Writable]]: true, 
   [[Enumerable]]: true, 
   [[Configurable]]: true
}
5. Return Property Identifier (propName, desc).

The PropName of the production `PropertyAssignment : get PropertyName ( ) { FunctionBody }` is determined as follows:

1. Return PropName(PropertyName).

The production `PropertyAssignment : get PropertyName ( ) { FunctionBody }` is evaluated as follows:

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with an empty parameter list and body specified by FunctionBody. Pass in the LexicalEnvironment of the running execution context as the Scope. Pass in `true` as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.
3. Let desc be the Property Descriptor {
   [[Get]]: closure, 
   [[Enumerable]]: true, 
   [[Configurable]]: true
}
4. Return Property Identifier (propName, desc).

The PropName of the production `PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }` is determined as follows:

1. Return PropName(PropertyName).

The static semantics of the production `PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }` are:

- It is a Syntax Error if the source code matching this production is extended code and the PropName of PropertyName also occurs in the VarDeclaredNames of FunctionBody.
- It is a Syntax Error if the source code matching this production is extended code and the PropName of PropertyName also occurs in the LexicallyDeclaredNames of FunctionBody.
- It is a Syntax Error if the source code matching this production is extended code and any element of the LexicallyDeclaredNames of PropertySetParameterList also occurs in the VarDeclaredNames of FunctionBody.
- It is a Syntax Error if the source code matching this production is extended code and any element of the BoundNames of PropertySetParameterList also occurs in the LexicallyDeclaredNames of FunctionBody.

The production `PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }` is evaluated as follows:
1. Let \( \textit{propName} \) be \( \text{PropName}(\textit{PropertyName}) \).
2. Let \( \textit{closure} \) be the result of creating a new Function object as specified in 13.2 with parameters specified by \( \text{PropertySetParameterList} \) and body specified by \( \text{FunctionBody} \). Pass in the LexicalEnvironment of the running execution context as the \( \textit{Scope} \). Pass in \( \text{true} \) as the \( \textit{Strict} \) flag if the \( \text{PropertyAssignment} \) is contained in strict code or if its \( \text{FunctionBody} \) is strict code.
3. Let \( \textit{desc} \) be the Property Descriptor\{[[Set]]: \( \textit{closure} \), [[Enumerable]]: \( \text{true} \), [[Configurable]]: \( \text{true} \)\}
4. Return Property Identifier (\( \textit{propName}, \textit{desc} \)).

The PropName of the production \( \textit{PropertyName} : \textit{IdentifierName} \) is evaluated as follows:

1. Return \( \text{PropName}(\textit{IdentifierName}) \).

The PropName of the production \( \textit{PropertyName} : \textit{stringLiteral} \) is evaluated as follows:

1. Return the SV of the \( \textit{stringLiteral} \).

The PropName of the production \( \textit{PropertyName} : \textit{numericLiteral} \) is evaluated as follows:

1. Let \( \textit{nbr} \) be the result of forming the value of the \( \textit{numericLiteral} \).
2. Return \( \text{toString}(\textit{nbr}) \).

The PropName of the token \( \textit{IdentifierName} \) is determined as follows:

1. Return the String value containing the same sequence of characters as \( \textit{IdentifierName} \).

The ExpectedArgumentCount of the production \( \textit{PropertySetParameterList} : \textit{BindingIdentifier} \) is determined as follows:

1. Return 1.

HasInitialiser of the production \( \textit{PropertySetParameterList} : \textit{BindingIdentifier} \) is determined as follows:

1. Return \( \text{false} \).

The BoundNames of the production \( \textit{PropertySetParameterList} : \textit{BindingIdentifier} \) is determined as follows:

1. Return BoundNames of \( \textit{BindingIdentifier} \).

The static semantics of the production \( \textit{PropertySetParameterList} : \textit{BindingPattern} \) are:

- It is a Syntax Error if the source code parsed with this production is not extended code.
- It is a Syntax Error if BoundNames of \( \textit{BindingPattern} \) contains any duplicate elements.

The ExpectedArgumentCount of the production \( \textit{PropertySetParameterList} : \textit{BindingPattern} \) is determined as follows:

1. Return 1.

HasInitialiser of the production \( \textit{PropertySetParameterList} : \textit{BindingPattern} \) is determined as follows:

1. Return \( \text{false} \).

The BoundNames of the production \( \textit{PropertySetParameterList} : \textit{BindingPattern} \) is determined as follows:

1. Return BoundNames of \( \textit{BindingPattern} \).
The following is an experimental alternative presentation of the above section

In this version the semantic definitions for all productions defined in this section are grouped together by semantic term. Static Semantics (if any) is always first and evaluation semantics is always last.

### 11.1.5 Object Initialiser

#### NOTE

An object initialiser is an expression describing the initialisation of an Object, written in a form resembling a literal. It is a list of zero or more pairs of property names and associated values, enclosed in curly braces. The values need not be literals; they are evaluated each time the object initialiser is evaluated.

#### Syntax

ObjectLiteral:

```plaintext
  { }
  { PropertyNameAndValueList }
  { PropertyNameAndValueList , }
```

PropertyNameAndValueList:

- `PropertyNameAssignment`
- `PropertyNameAndValueList , PropertyAssignment`

PropertyAssignment:

- `IdentifierName`
- `PropertyName : AssignmentExpression`
- `PropertyName ( FormalParameterList ) { FunctionBody }`
- `get PropertyName ( ) { FunctionBody }`
- `set PropertyName ( PropertySetParameterList ) { FunctionBody }`

PropertyName:

- `IdentifierName`
- `StringLiteral`
- `NumericLiteral`

PropertySetParameterList:

- `BindingIdentifier`
- `BindingPattern`

#### Static Semantics

PropertyNameAndValueList:

- `PropertyNameAndValueList, PropertyAssignment`

- It is a Syntax Error if this production is contained in strict code, `PropertyAssignment` is the production `PropertyAssignment : PropertyName : AssignmentExpression`, and `PropertyDefinitionList(PropName of PropertyAssignment) of PropertyNameAndValueList` is not the empty List.

- It is a Syntax Error if `PropertyAssignment` is the production `PropertyAssignment : get PropertyName ( ) { FunctionBody }` and `PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList` includes a production of the form `PropertyAssignment : PropertyName : AssignmentExpression`.

- It is a Syntax Error if `PropertyAssignment` is the production `PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }` and `PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList` includes a production of the form `PropertyAssignment : PropertyName : AssignmentExpression`.

---

© Ecma International 2011
• It is a Syntax Error if `PropertyAssignment` is the production
  
  ```
  PropertyAssignment : get PropertyName ( ) { FunctionBody }
  ```
  and PropertyDefinitionList (PropName of `PropertyAssignment`) of `PropertyNameAndValueList` includes a production of the form
  ```
  PropertyAssignment : get PropertyName ( ) { FunctionBody }.
  ```

• It is a Syntax Error if `PropertyAssignment` is the production
  ```
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }
  ```
  and PropertyDefinitionList (PropName of `PropertyAssignment`) of `PropertyNameAndValueList` includes a production of the form
  ```
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }.
  ```

`PropertyAssignment`: `get` `PropertyName` ( `PropertySetParameterList` ) { `FunctionBody` }

• It is a Syntax Error if the source code matching this production is extended code and the PropName of `PropertyName` also occurs in the `VarDeclaredNames` of `FunctionBody`.

• It is a Syntax Error if the source code matching this production is extended code and the PropName of `PropertyName` also occurs in the `LexicallyDeclaredNames` of `FunctionBody`.

• It is a Syntax Error if the source code matching this production is extended code and any element of the `LexicallyDeclaredNames` of `PropertySetParameterList` also occurs in the `VarDeclaredNames` of `FunctionBody`.

• It is a Syntax Error if the source code matching this production is extended code and any element of the `BoundNames` of `PropertySetParameterList` also occurs in the `LexicallyDeclaredNames` of `FunctionBody`.

`PropertySetParameterList`: `BindingPattern`

• It is a Syntax Error if the source code parsed with this production is not extended code.

• It is a Syntax Error if `BoundNames` of `BindingPattern` contains any duplicate elements.

**Semantics:** `PropertyDefinitionList(name)`

`PropertyAssignment`: `PropertyName`: `AssignmentExpression`

1. If PropName of `PropertyAssignment` is not `name` return the empty List.
2. Return a List containing `PropertyAssignment`.

`PropertyNameAndValueList`: `PropertyNameAndValueList`, `PropertyAssignment`

1. Let `previous` be `PropertyDefinitionList(name)` of `PropertyNameAndValueList`.
2. If PropName of `PropertyAssignment` is `name` then,
   a. Append `PropertyAssignment` to the end of `previous`.
3. Return previous.

**Semantics:** PropName

`PropertyAssignment`: `IdentifierName`

1. Return PropName(IdentifierName).

`PropertyAssignment`: `PropertyName`: `AssignmentExpression`

1. Return PropName(`PropertyName`).

`PropertyAssignment`: `get` `PropertyName` ( ) { `FunctionBody` }

1. Return PropName(`PropertyName`).

`PropertyAssignment`: `set` `PropertyName` ( `PropertySetParameterList` ) { `FunctionBody` }
1. Return PropName(Property).  

PROPERTYNAME : IDENTIFIERNAME
1. Return PropName(IdentifierName).  

PROPERTYNAME : STRINGLITERAL
1. Return the SV of the StringLiteral.  

PROPERTYNAME : NUMERICLITERAL
1. Let \( nbr \) be the result of forming the value of the NUMERICLITERAL.  
2. Return ToString(nbr).

The token IDENTIFIERNAME
1. Return the String value containing the same sequence of characters as IDENTIFIERNAME.

**Semantics:** ExpectedArgumentCount

PROPERTYSETPARAMETERLIST : BINDINGIDENTIFIER
1. Return 1.

PROPERTYSETPARAMETERLIST : BINDINGPATTERN
1. Return 1.

**Semantics:** HasInitialiser

PROPERTYSETPARAMETERLIST : BINDINGIDENTIFIER
1. Return false.

PROPERTYSETPARAMETERLIST : BINDINGPATTERN
1. Return false.

**Semantics:** BoundNames

PROPERTYSETPARAMETERLIST : BINDINGIDENTIFIER
1. Return BoundNames of BINDINGIDENTIFIER.

PROPERTYSETPARAMETERLIST : BINDINGPATTERN
1. Return BoundNames of BINDINGPATTERN.

**Semantics:** Evaluation

OBJECTLITERAL : \{ \}
1. Return a new object created as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
ObjectLiteral: { PropertyNameAndValueList }

ObjectLiteral: { PropertyNameAndValueList , }

1. Return the result of evaluating PropertyNameAndValueList.

PropertyNameAndValueList: PropertyAssignment

1. Let obj be the result of creating a new object as if by the expression new Object() where Object is the standard built-in constructor with that name.
2. Let propId be the result of evaluating PropertyAssignment.
3. Call the [[DefineOwnProperty]] internal method of obj with arguments propId.name, propId.descriptor, and false.
4. Return obj.

PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment

1. Let obj be the result of evaluating PropertyNameAndValueList.
2. Let propId be the result of evaluating PropertyAssignment.
3. Call the [[DefineOwnProperty]] internal method of obj with arguments propId.name, propId.descriptor, and false.
4. Return obj.

PropertyAssignment: IdentifierName

1. Let propName be PropName(IdentifierName).
2. Let exprValue be the result of performing Identifier Resolution as specified in 10.3.1 using IdentifierName.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor{[[Value]]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}
5. Return Property Identifier (propName, desc).

PropertyAssignment: PropertyName : AssignmentExpression

1. Let propName be PropName(PropertyName).
2. Let exprValue be the result of evaluating AssignmentExpression.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor{[[Value]]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}
5. Return Property Identifier (propName, desc).

PropertyAssignment: get PropertyName ( ) { FunctionBody }

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with an empty parameter list and body specified by FunctionBody. Pass in the LexicalEnvironment of the running execution context as the Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.
3. Let desc be the Property Descriptor{[[Get]]: closure, [[Enumerable]]: true, [[Configurable]]: true}
4. Return Property Identifier (propName, desc).

PropertyAssignment: set PropertyName ( PropertySetParameterList ) { FunctionBody }

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with parameters specified by PropertySetParameterList and body specified by FunctionBody. Pass in the LexicalEnvironment of the running execution context as the Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.
3. Let desc be the Property Descriptor{[[Set]]: closure, [[Enumerable]]: true, [[Configurable]]: true}
4. Return Property Identifier (propName, desc).
#3 The following is an experimental alternative presentation of the above section

This version is just like the previous experiment except each semantic term group has a numbered subsection that shows up in the table of contents/navigation pane.

11.1.5 Object Initialiser

NOTE  An object initialiser is an expression describing the initialisation of an Object, written in a form resembling a literal. It is a list of zero or more pairs of property names and associated values, enclosed in curly braces. The values need not be literals; they are evaluated each time the object initialiser is evaluated.

11.1.5.1 Syntax

ObjectLiteral :  
{ }  
{ PropertyNameAndValueList , }  

PropertyNameAndValueList :  
PropertyNameAssignment  
PropertyNameAndValueList , PropertyAssignment

PropertyAssignment :  
PropertyName : AssignmentExpression  
PropertyName ( FormalParameterListopt ) { FunctionBody }  
get PropertyName ( ) { FunctionBody }  
set PropertyName ( PropertySetParameterList ) { FunctionBody }

PropertyName :  
IdentifierName  
StringLiteral  
NumericLiteral

PropertySetParameterList :  
BindingIdentifier  
BindingPattern

11.1.5.2 Static Semantics

PropertyNameAndValueList : PropertyNameAndValueList , PropertyAssignment

• It is a Syntax Error if this production is contained in strict code, PropertyAssignment is the production PropertyAssignment : PropertyName : AssignmentExpression, and PropertyDefinitionList(PropName of PropertyAssignment) of PropertyNameAndValueList is not the empty List.
• It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : get PropertyName ( ) { FunctionBody } and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : PropertyName : AssignmentExpression.
• It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody } and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : PropertyName : AssignmentExpression.
• It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : get PropertyName ( ) { FunctionBody }
and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a
production of the form PropertyAssignment : get PropertyName ( ) { FunctionBody }.

• It is a Syntax Error if PropertyAssignment is the production
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }
and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a
production of the form
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }.

PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }

• It is a Syntax Error if the source code matching this production is extended code and the PropName of
  PropertyName also occurs in the VarDeclaredNames of FunctionBody.
• It is a Syntax Error if the source code matching this production is extended code and the PropName of
  PropertyName also occurs in the LexicallyDeclaredNames of FunctionBody.
• It is a Syntax Error if the source code matching this production is extended code and any element of
  the LexicallyDeclaredNames of PropertySetParameterList also occurs in the VarDeclaredNames of
  FunctionBody.
• It is a Syntax Error if the source code matching this production is extended code and any element of
  the BoundNames of PropertySetParameterList also occurs in the LexicallyDeclaredNames of
  FunctionBody.

PropertySetParameterList : BindingPattern

• It is a Syntax Error if the source code parsed with this production is not extended code.
• It is a Syntax Error if BoundNames of BindingPattern contains any duplicate elements.

11.1.5.3 Semantics: PropertyDefinitionList(name)

PropertyAssignment : PropertyName : AssignmentExpression

1. If PropName of PropertyAssignment is not name return the empty List.
2. Return a List containing PropertyAssignment.

PropertyNameAndValueList : PropertyNameAndValueList , PropertyAssignment

1. Let previous be PropertyDefinitionList(name) of PropertyNameAndValueList.
2. If PropName of PropertyAssignment is name then,
   a. Append PropertyAssignment to the end of previous.
3. Return previous.

11.1.5.4 Semantics: PropName

PropertyAssignment : IdentifierName

1. Return PropName(IdentifierName).

PropertyAssignment : PropertyName : AssignmentExpression

1. Return PropName(PropertyName).

PropertyAssignment : get PropertyName ( ) { FunctionBody }

1. Return PropName(PropertyName).

PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }
1. Return PropName(PropertyName).

PropertyName : IdentifierName

1. Return PropName(IdentifierName).

PropertyName : StringLiteral

1. Return the SV of the StringLiteral.

PropertyName : NumericLiteral

1. Let nbr be the result of forming the value of the NumericLiteral.
2. Return ToString(nbr).

The token IdentifierName

1. Return the String value containing the same sequence of characters as IdentifierName.

11.1.5.5 Semantics: ExpectedArgumentCount

PropertySetParameterList : BindingIdentifier

1. Return 1.

PropertySetParameterList : BindingPattern

1. Return 1.

11.1.5.6 Semantics: HasInitialiser

PropertySetParameterList : BindingIdentifier

1. Return false.

PropertySetParameterList : BindingPattern

1. Return false.

11.1.5.7 Semantics: BoundNames

PropertySetParameterList : BindingIdentifier

1. Return BoundNames of BindingIdentifier.

PropertySetParameterList : BindingPattern

1. Return BoundNames of BindingPattern.

11.1.5.8 Semantics: Evaluation

ObjectLiteral : { }

1. Return a new object created as if by the expression new Object() where Object is the standard built-in constructor with that name.

ObjectLiteral : { PropertyNameAndValueList }
ObjectLiteral : { PropertyNameAndValueList , }  

1. Return the result of evaluating PropertyNameAndValueList.

PropertyNameAndValueList : PropertyAssignment

1. Let obj be the result of creating a new object as if by the expression new Object() where Object is the standard built-in constructor with that name.
2. Let propId be the result of evaluating PropertyAssignment.
3. Call the [[DefineOwnProperty]] internal method of obj with arguments propId.name, propId.descriptor, and false.
4. Return obj.

PropertyNameAndValueList : PropertyNameAndValueList , PropertyAssignment

1. Let obj be the result of evaluating PropertyNameAndValueList.
2. Let propId be the result of evaluating PropertyAssignment.
3. Call the [[DefineOwnProperty]] internal method of obj with arguments propId.name, propId.descriptor, and false.
4. Return obj.

PropertyAssignment : IdentifierName

1. Let propName be PropName(IdentifierName).
2. Let exprValue be the result of performing Identifier Resolution as specified in 10.3.1 using IdentifierName.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor{[[Value]]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}
5. Return Property Identifier (propName, desc).

PropertyAssignment : PropertyName : AssignmentExpression

1. Let propName be PropName(PropertyName).
2. Let exprValue be the result of evaluating AssignmentExpression.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor{[[Value]]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}
5. Return Property Identifier (propName, desc).

PropertyAssignment : get PropertyName ( ) { FunctionBody

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with an empty parameter list and body specified by FunctionBody. Pass in the LexicalEnvironment of the running execution context as the Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.
3. Let desc be the Property Descriptor{[[Get]]: closure, [[Enumerable]]: true, [[Configurable]]: true}
4. Return Property Identifier (propName, desc).

PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with parameters specified by PropertySetParameterList and body specified by FunctionBody. Pass in the LexicalEnvironment of the
running execution context as the Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.

3. Let desc be the Property Descriptor {[[Set]]: closure, [[Enumerable]]: true, [[Configurable]]: true}

4. Return Property Identifier (propName, desc).
The following is an experimental alternative presentation of the above section

This version groups semantic definitions by the production that they relate to and labels each group with its production. It is most similar to the first presentation that is in the current draft.

11.1.5 Object Initialiser

NOTE An object initialiser is an expression describing the initialisation of an Object, written in a form resembling a literal. It is a list of zero or more pairs of property names and associated values, enclosed in curly braces. The values need not be literals; they are evaluated each time the object initialiser is evaluated.

Syntax

ObjectLiteral :
  { } 
  { PropertyNameAndValueList } 
  { PropertyNameAndValueList , }

PropertyNameAndValueList :
  PropertyAssignment 
  PropertyNameAndValueList , PropertyAssignment

PropertyAssignment :
  IdentifierName 
 (PropertyName : AssignmentExpression 
  PropertyName ( FormalParameterListopt ) { FunctionBody } 
  get PropertyName ( ) { FunctionBody } 
  set PropertyName ( PropertySetParameterList ) { FunctionBody }

PropertyName :
  IdentifierName 
  StringLiteral 
  NumericLiteral

PropertySetParameterList :
  BindingIdentifier 
  BindingPattern

Semantics

ObjectLiteral :
  { } 

The production is evaluated as follows:

1. Return a new object created as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.

ObjectLiteral : { PropertyNameAndValueList } 
ObjectLiteral : { PropertyNameAndValueList , }

The production is evaluated as follows:

1. Return the result of evaluating `PropertyNameAndValueList`. 
PropertyAssignment : PropertyName : AssignmentExpression

The PropertyDefinitionList(name) of the production is determined as follows:

1. If PropName of PropertyAssignment is not name return the empty List.
2. Return a List containing PropertyAssignment.

PropertyNameAndValueList : PropertyAssignment

The production is evaluated as follows:

1. Let obj be the result of creating a new object as if by the expression new Object() where Object is the standard built-in constructor with that name.
2. Let propId be the result of evaluating PropertyAssignment.
3. Call the [[DefineOwnProperty]] internal method of obj with arguments propId.name, propId.descriptor, and false.
4. Return obj.

PropertyNameAndValueList : PropertyNameAndValueList , PropertyAssignment

The static semantics are:

- It is a Syntax Error if this production is contained in strict code, PropertyAssignment is the production PropertyAssignment : PropertyName : AssignmentExpression, and PropertyDefinitionList(PropName of PropertyAssignment) of PropertyNameAndValueList is not the empty List.
- It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : get PropertyName ( ) { FunctionBody } and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : PropertyName : AssignmentExpression.
- It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody } and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : get PropertyName ( ) { FunctionBody }.
- It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody } and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }.

PropertyDefinitionList(name) of the production is determined as follows:

1. Let previous be PropertyDefinitionList(name) of PropertyNameAndValueList.
2. If PropName of PropertyAssignment is name then,
   b. Append PropertyAssignment to the end of previous.
3. Return previous.

The production is evaluated as follows:

1. Let obj be the result of evaluating PropertyNameAndValueList.
2. Let propId be the result of evaluating PropertyAssignment.
3. Call the [[DefineOwnProperty]] internal method of obj with arguments propId.name, propId.descriptor, and false.
4. Return obj.

© Ecma International 2009 – All rights reserved
PropertyAssignment : IdentifierName

PropName of the production is determined as follows:

1. Return PropName(IdentifierName).

The production is evaluated as follows:

1. Let propName be PropName(IdentifierName).
2. Let exprValue be the result of performing Identifier Resolution as specified in 10.3.1 using IdentifierName.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor {{Value]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}
5. Return Property Identifier (propName, desc).

PropertyAssignment : PropertyName : AssignmentExpression

PropName of the production is determined as follows:

1. Return PropName(PropertyName).

The production is evaluated as follows:

1. Let propName be PropName(PropertyName).
2. Let exprValue be the result of evaluating AssignmentExpression.
3. Let propValue be GetValue(exprValue).
4. Let desc be the Property Descriptor {{Value]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}
5. Return Property Identifier (propName, desc).

PropertyAssignment : get PropertyName ( ) { FunctionBody }

PropName of the production is determined as follows:

1. Return PropName(PropertyName).

The production is evaluated as follows:

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with an empty parameter list and body specified by FunctionBody. Pass in the LexicalEnvironment of the running execution context as the Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.
3. Let desc be the Property Descriptor {{[Get]]: closure, [[Enumerable]]: true, [[Configurable]]: true}
4. Return Property Identifier (propName, desc).

PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }

The static semantics are:

- It is a Syntax Error if the source code matching this production is extended code and the PropName of PropertyName also occurs in the VarDeclaredNames of FunctionBody.
- It is a Syntax Error if the source code matching this production is extended code and the PropName of PropertyName also occurs in the LexicallyDeclaredNames of FunctionBody.
• It is a Syntax Error if the source code matching this production is extended code and any element of the LexicallyDeclaredNames of `PropertySetParameterList` also occurs in the VarDeclaredNames of `FunctionBody`.
• It is a Syntax Error if the source code matching this production is extended code and any element of the BoundNames of `PropertySetParameterList` also occurs in the LexicallyDeclaredNames of `FunctionBody`.

**PropName** of the production is determined as follows:

1. Return `PropName(PropertyName)`.

The production is evaluated as follows:

1. Let `propName` be `PropName(PropertyName)`.
2. Let `closure` be the result of creating a new Function object as specified in 13.2 with parameters specified by `PropertySetParameterList` and body specified by `FunctionBody`. Pass in the LexicalEnvironment of the running execution context as the `Scope`. Pass in `true` as the `Strict` flag if the `PropertyAssignment` is contained in strict code or if its `FunctionBody` is strict code.
3. Let `desc` be the Property Descriptor{[[Set]]: `closure`, [[Enumerable]]: `true`, [[Configurable]]: `true`}
4. Return Property Identifier (`propName`, `desc`).

**PropertyName** : `IdentifierName`

**PropName** of the production is determined as follows:

1. Return `PropName(IdentifierName)`.

**PropertyName** : `StringLiteral`

**PropName** of the production is determined as follows:

1. Return the SV of the `StringLiteral`.

**PropertyName** : `NumericLiteral`

**PropName** of the production is determined as follows:

1. Let `nbr` be the result of forming the value of the `NumericLiteral`.
2. Return `ToString(nbr)`.

**IdentifierName**

**PropName** of the token is determined as follows:

1. Return the String value containing the same sequence of characters as `IdentifierName`.

**PropertySetParameterList** : `BindingIdentifier`

ExpectedArgumentCount of the production is determined as follows:

1. Return 1.

**HasInitialiser** of the production is determined as follows:

1. Return `false`.

**BoundNames** of the production is determined as follows:

1. Return BoundNames of `BindingIdentifier`.
PropertySetParameterList : BindingPattern

The static semantics are:

- It is a Syntax Error if the source code parsed with this production is not extended code.
- It is a Syntax Error if BoundNames of BindingPattern contains any duplicate elements.

ExpectedArgumentCount of the production is determined as follows:

1. Return 1.

HasInitialiser of the production is determined as follows:

1. Return false.

BoundNames of the production is determined as follows:

1. Return BoundNames of BindingPattern.
#5 The following is an experimental alternative presentation of this section

This version is just like experiment 3 except that there are only three numbered subsections, syntax, static semantics, and runtime semantics. Bold subheadings are used to identify each group of related definitions within a subsection. Semantic functions that only depend upon static characteristics of the program text are placed in the static semantics section.

11.1.5 Object Initialiser

NOTE An object initialiser is an expression describing the initialisation of an Object, written in a form resembling a literal. It is a list of zero or more pairs of property names and associated values, enclosed in curly braces. The values need not be literals; they are evaluated each time the object initialiser is evaluated.

11.1.5.1 Syntax

ObjectLiteral:
    { }
    { PropertyNameAndValueList }
    { PropertyNameAndValueList , }

PropertyNameAndValueList:
    PropertyAssignment
    PropertyNameAndValueList , PropertyAssignment

PropertyAssignment:
    IdentifierName
    PropertyName : AssignmentExpression
   (PropertyName ( FormalParameterListopt ) ) { FunctionBody }
    get PropertyName ( ) { FunctionBody }
    set PropertyName ( PropertySetParameterList ) { FunctionBody }

PropertyName:
    IdentifierName
    StringLiteral
    NumericLiteral

PropertySetParameterList:
    BindingIdentifier
    BindingPattern

11.1.5.2 Static Semantics

Early Errors

PropertyNameAndValueList:
    PropertyNameAndValueList , PropertyAssignment

- It is a Syntax Error if this production is contained in strict code, PropertyAssignment is the production PropertyAssignment : PropertyName : AssignmentExpression, and PropertyDefinitionList(PropName of PropertyAssignment) of PropertyNameAndValueList is not the empty List.
- It is a Syntax Error if PropertyAssignment is the production PropertyAssignment : get PropertyName ( ) { FunctionBody }
  and PropertyDefinitionList(PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : PropertyName : AssignmentExpression.
- It is a Syntax Error if PropertyAssignment is the production
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }
and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : PropertyName : AssignmentExpression.

- It is a Syntax Error if PropertyAssignment is the production
  PropertyAssignment : get PropertyName ( ) { FunctionBody }.

and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form PropertyAssignment : get PropertyName ( ) { FunctionBody }.

- It is a Syntax Error if PropertyAssignment is the production
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }.

and PropertyDefinitionList (PropName of PropertyAssignment) of PropertyNameAndValueList includes a production of the form
  PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }.

PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }

- It is a Syntax Error if the source code matching this production is extended code and the PropName of PropertyName also occurs in the VarDeclaredNames of FunctionBody.

- It is a Syntax Error if the source code matching this production is extended code and the PropName of PropertyName also occurs in the LexicallyDeclaredNames of FunctionBody.

- It is a Syntax Error if the source code matching this production is extended code and any element of the LexicallyDeclaredNames of PropertySetParameterList also occurs in the VarDeclaredNames of FunctionBody.

- It is a Syntax Error if the source code matching this production is extended code and any element of the BoundNames of PropertySetParameterList also occurs in the LexicallyDeclaredNames of FunctionBody.

PropertySetParameterList : BindingPattern

- It is a Syntax Error if the source code parsed with this production is not extended code.

- It is a Syntax Error if BoundNames of BindingPattern contains any duplicate elements.

**Static Semantics: PropertyDefinitionList(name)**

PropertyAssignment : PropertyName : AssignmentExpression

1. If PropName of PropertyAssignment is not name return the empty List.
2. Return a List containing PropertyAssignment.

PropertyNameAndValueList : PropertyNameAndValueList , PropertyAssignment

1. Let previous be PropertyDefinitionList(name) of PropertyNameAndValueList.
2. If PropName of PropertyAssignment is name then,
   a. Append PropertyAssignment to the end of previous.
3. Return previous.

**Static Semantics: PropName**

PropertyAssignment : IdentifierName

1. Return PropName(IdentifierName).

PropertyAssignment : PropertyName : AssignmentExpression

1. Return PropName(PropertyName).
PropertyAssignment : \( \text{get } \) PropertyName \( (\quad) \) \{ FunctionBody \}

1. Return PropName(PropertyName).

PropertyAssignment : \( \text{set } \) PropertyName \( (\) PropertySetParameterList \( ) \) \{ FunctionBody \}

1. Return PropName(PropertyName).

PropertyName : IdentifierName

1. Return PropName(IdentifierName).

PropertyName : StringLiteral

1. Return the SV of the StringLiteral.

PropertyName : NumericLiteral

1. Let \( nbr \) be the result of forming the value of the NumericLiteral.
2. Return ToString(nbr).

The token IdentifierName

1. Return the String value containing the same sequence of characters as IdentifierName.

**Static Semantics: ExpectedArgumentCount**

PropertySetParameterList : BindingIdentifier

1. Return 1.

PropertySetParameterList : BindingPattern

1. Return 1.

**Static Semantics: HasInitialiser**

PropertySetParameterList : BindingIdentifier

1. Return false.

PropertySetParameterList : BindingPattern

1. Return false.

**Static Semantics: BoundNames**

PropertySetParameterList : BindingIdentifier

1. Return BoundNames of BindingIdentifier.

PropertySetParameterList : BindingPattern

1. Return BoundNames of BindingPattern.
11.1.5.3 Runtime Semantics

Runtime Semantics: Evaluation

ObjectLiteral : { }

1. Return a new object created as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.

ObjectLiteral : { PropertyNameAndValueList }

ObjectLiteral : { PropertyNameAndValueList , }

1. Return the result of evaluating `PropertyNameAndValueList`.

PropertyNameAndValueList : PropertyAssignment

1. Let `obj` be the result of creating a new object as if by the expression `new Object()` where `Object` is the standard built-in constructor with that name.
2. Let `propId` be the result of evaluating `PropertyAssignment`.
3. Call the `[[DefineOwnProperty]]` internal method of `obj` with arguments `propId.name`, `propId.descriptor`, and `false`.
4. Return `obj`.

PropertyNameAndValueList : PropertyNameAndValueList , PropertyAssignment

1. Let `obj` be the result of evaluating `PropertyNameAndValueList`.
2. Let `propId` be the result of evaluating `PropertyAssignment`.
3. Call the `[[DefineOwnProperty]]` internal method of `obj` with arguments `propId.name`, `propId.descriptor`, and `false`.
4. Return `obj`.

PropertyAssignment : IdentifierName

1. Let `propName` be `PropName(IdentifierName)`.
2. Let `exprValue` be the result of performing Identifier Resolution as specified in 10.3.1 using `IdentifierName`.
3. Let `propValue` be `GetValue(exprValue)`.
4. Let `desc` be the Property Descriptor `{{[Value]]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}`
5. Return Property Identifier `(propName, desc)`.

PropertyAssignment : PropertyName : AssignmentExpression

1. Let `propName` be `PropName(PropertyName)`.
2. Let `exprValue` be the result of evaluating `AssignmentExpression`.
3. Let `propValue` be `GetValue(exprValue)`.
4. Let `desc` be the Property Descriptor `{{[Value]]: propValue, [[Writable]]: true, [[Enumerable]]: true, [[Configurable]]: true}`
5. Return Property Identifier `(propName, desc)`.

PropertyAssignment : `get` PropertyName ( ) { FunctionBody

1. Let `propName` be `PropName(PropertyName)`.
2. Let `closure` be the result of creating a new Function object as specified in 13.2 with an empty parameter list and body specified by `FunctionBody`. Pass in the LexicalEnvironment of the running execution context as the
Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.

3. Let desc be the Property Descriptor \{[[Get]]: closure, [[Enumerable]]: true, [[Configurable]]: true\}

4. Return Property Identifier (propName, desc).

PropertyAssignment : set PropertyName ( PropertySetParameterList ) { FunctionBody }

1. Let propName be PropName(PropertyName).
2. Let closure be the result of creating a new Function object as specified in 13.2 with parameters specified by PropertySetParameterList and body specified by FunctionBody. Pass in the LexicalEnvironment of the running execution context as the Scope. Pass in true as the Strict flag if the PropertyAssignment is contained in strict code or if its FunctionBody is strict code.

3. Let desc be the Property Descriptor \{[[Set]]: closure, [[Enumerable]]: true, [[Configurable]]: true\}

4. Return Property Identifier (propName, desc).