# Prog2 GHa

Generated by Doxygen 1.8.14

# Contents

# Chapter 1

# Große Hausaufgabe für Prog2

Benutzerdokumentation

### Build

Um die Projekt zu bilden, muss man diese Repository klonieren (z.B. von `GitHub` herunterladen), und eine einzige `make` Befehl ausgeben im Root des Projektes. Diese wird zwei Foldern herstellen, `obj` und `build`, welche jeweils die .o Objektdateien und die hergestellte und laufbare Programm enthalten.

### Laufen lassen

Diese Programm lässt sich mit Hilfe von CLI Switches laufen lassen. Diese sind:

- `-p`: Eine zu füllende Sprachendatei ausschreiben in den folgenden Datei (z.B. `./prog2 -p language.lang` wird eine Sample im File `language.lang` ausschreiben). Falls diese CLI Switch benutzt wird, wird der Programm nach der Filegenerierung sich schließen.
- `-l`: Eine gefüllte Sprachendatei einladen und verarbeiten. Die Forme dieser Datei ist:

```
{
"Move B to A":"mov",
"Add B to A":"add",
"Substract B from A":"sub",
"Swap the upper and lower 4 bits":"swp",
"Shift left, insert 0":"sl0",
"Shift right, insert 0":"sr0",
"Jump to subroutine":"jsr",
"Push value to stack":"push",
"Pop value from stack":"pop"
}
```

- `-f`: Kann nur als die letzte CLI Switch benutzt werden. Wird die nächste Dateien als Subroutine einladen und verarbeiten (können später benutzt werden). Diese Dateien sehen so aus:

```
Name der Subroutine
<instruction>
<instruction>
...
```

  Die letzte zwei Switches sind nicht nötig, ohne sie kann die Programm auch laufen gelassen, dann werden die Defaultwerte benutzt.

## Instruktionen

Der VirtualMachine, der benutzt wird, ist eine 8-bit Gerät. Wegen Zeitzwang wurde nur einige Instruktionen implementiert (nämlich mov, add, sub, swp, sl0, sr0, jsr, push und pop), aber die Addition von anderen Instruktionen ist sehr einfach (Siehe die Programmiererdokumentation für weitere Informationen). Diese Instruktionen sind im Defaultsprachendatei:

```
mov A B: kopiert den Wert von B in A
add A B: addiert den Wert von B zu A
sub A B: substrahiert den Wert von B aus A
swp A: Austauscht den untere und obene 4 Bits
sl0 A: Verschiebt den Wert von A nach links mit 1 Bit
sr0 A: Verschiebt den Wert von A nach rechts mit 1 Bit
jsr <label>: Lässt den Instruktionen definiert im Subroutine "label" laufen
push B: Drückt den Wert von B zum Stack
pop A: Popt den Topwert aus dem Stack im A
```

Die Werte können sein:

- A: Register (r1-r16), speicher (Im Form (rN) addressiert)
- B: Register (r1-r16), speicher (Im Form (rN) addressiert) oder Literal (0xNN)

Spezielle Instruktionen:

- `quit`: Schließt den Programm
- `(r1)` - `(r16)`: Schreibt den Wert der Speicherbereich addressiert durch einen Register auf dem Bildschirm
- `r1` - `r16`: Schreibt den Wert der Register auf dem Bildschirm

## Ein Beispiel

Die exakte Operation des Programmes zu zeigen steht hier folgender Beispiel für den Benutz des Applikations:

1. Die Sprachendatei:

   Jede Instruktion wird mit einem anderen ersetzt. Die Sprachendatei steht hier:

   ```
   {
       "Move B to A":"ertek",
       "Add B to A":"hozzaad",
       "Substract B from A":"kivon",
       "Swap the upper and lower 4 bits":"csere",
       "Shift left, insert 0":"balra",
       "Shift right, insert 0":"jobbra",
       "Jump to subroutine":"ugras",
       "Push value to stack":"mentes",
       "Pop value from stack":"betoltes"
   }
   ```

2. Die Subroutinendatei

   ```
   sbr
   betoltes r1
   hozzaad r1 r1
   kivon r1 0x01
   csere r1
   balra r1
   jobbra r1
   mentes r1
   ```

3. Der Laufsparametern

   ```
   ./prog2 -l language.lang -f sbr
   ```

4. Die eingetippten Zeilen:

   ```
   ertek r10 0x50
   mentes r10
   r10
   ugras sbr
   betoltes r10
   r10
   ```

5. Die Ausgang:

   ```
   [levente@archlinux test]$ ./prog2 -l language.lang -f sbr
   ertek r10 0x50
   mentes r10
   r10
   Wert von r10: 0x50
   ugras sbr
   betoltes r10
   r10
   Wert von r10: 0x79
   quit
   ```

# Chapter 2

# Konzolbasiertes Interpreter eines dynamisch geschriebenen, konfigurierbaren Spraches

### Die Idee

Die Hauptidee des Programmes ist, den User eine eigene Beschreibungssprache konstruieren zu erlauben und dasselbe Sprache in einem dynamisch geschriebenen Art verwenden um eine eigene Programm zusammenzustellen (der später im File gespeichert werden kann). Das Sprache ist an den Prinzipen des Assembly-Spraches basiert, und enthält nur sehr atomische Instruktionen, die natürlich jede sinnvolle Problem lösen lassen.

### Die geplannte Umsetzung

Der Benutzer bekommt eine CLI Tool, die die folgende machen kann:

1. Eine leere, zu füllende Sprachendatei erstellen (JSON-basiert, damit es automatisch generiert werden kann sowohl aus dieser Programm als auch mit Hilfe von 3rd Party Programms).[1]
2. Eine gefüllte Sprachendatei einladen (und natürlich verifizieren, ob es wirklich gut gefüllt wurde).
3. Aus der eingeladeten Sprache einen inneren Konzol erstellen, in denen der Benutzer folgende tun kann:
   (a) Schritt für Schritt eine Programm eintippen und diese Schritte sofort laufen lassen.
   (b) Subroutine und Labels definieren (mit Hilfe von einer reservierten Schlusswort oder Bezeichnung, z.B. `def` oder `:`).
   (c) Definitionsbibliotheken aus speziellen Dateien einladen.
   (d) Soweit eingetippte Schritte ansehen und die ganze History im Datei speichern.
   (e) Registern, Stack und Speicher im File (oder aus dem Standardausgang) schreiben
4. Mit Hilfe von verschiedenen Sprachendateien Programme zwischen Sprachen umwandeln (erledigt z.B. wirkliche Assemblykode zu generieren, der später assembliert werden kann).

### Der Komplexität

Obwohl ich nach eine sehr einfaches Implementation streben möchte, damit es nicht schwierig wird den ganzen Software überzusehen, ist es unvermeidlich einige komplexere Sachen im Program zu verwenden. Damit zu hilfen habe ich die folgende Typusgraphen erstellt:

A VirtualMachine class für den Ausführung des Kodes:

Dieses VirtualMachine enthält verschiedene Speichern (Stack, Registers, RAM), diese sind in den folgenden Klassen implementiert:

Sowohl Instruktionen, als auch Sprachendateien können im File geschrieben werden, bzw. mit Dateioperationen ist eine JSON Implementation zu hilfen:

Diese sind natürlich nur Planen, und können während der Entwicklung geändert werden, sie stehen hier nur um die Erklärung der Idee zu erleitern.

Die folgende Komplexitätskriterien stehen im Plan (aus der 5 spezifizierte Kriterien):

- Templates
- Operatorüberladung
- Mehrfachvererbung
- Dynamische Membervariable

[1]: Eine Beispieldatei (nur die Idee zu zeigen, dass es eine eineindeutige Zuordnung von Mnemoniken zu inner definierte Funktionen enthält):

# Chapter 3

# Namespace Index

## 3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 4

# Hierarchical Index

## 4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 5

# Data Structure Index

## 5.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 6

# File Index

## 6.1 File List

Here is a list of all files with brief descriptions:

# Chapter 7

# Namespace Documentation

## 7.1 json Namespace Reference

### Namespaces

- anonymous_namespace{json.hpp}

### Data Structures

- class JSON

### Functions

- JSON Array ()
- template<typename... T>
  JSON Array (T... args)
- JSON Object ()
- std::ostream & operator<< (std::ostream &os, const JSON &json)

### 7.1.1 Function Documentation

#### 7.1.1.1 Array() [1/2]

```
JSON json::Array ( )  [inline]
```

Definition at line 421 of file json.hpp.

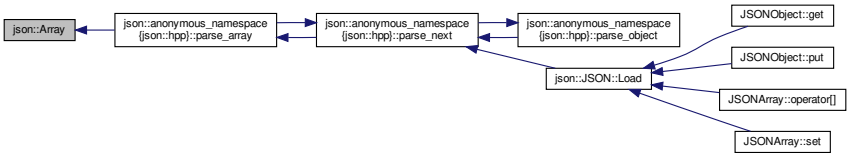
References json::JSON::Array, and json::JSON::Make().

Referenced by json::anonymous_namespace{json.hpp}::parse_array().

```
421                     {
422     return std::move( JSON::Make( JSON::Class::Array ) );
423 }
```

Here is the call graph for this function:


Here is the caller graph for this function:




#### 7.1.1.2 Array() [2/2]

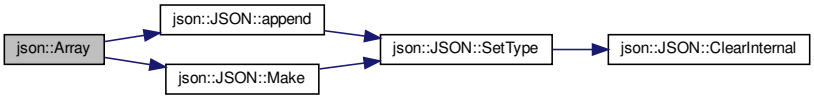
```
template<typename...  T>
JSON json::Array (
             T...  args )
```

Definition at line 426 of file json.hpp.

References json::JSON::append(), json::JSON::Array, and json::JSON::Make().

```
426                         {
427     JSON arr = JSON::Make( JSON::Class::Array );
428     arr.append( args... );
429     return std::move( arr );
430 }
```

Here is the call graph for this function:



#### 7.1.1.3 Object()

```
JSON json::Object ( )  [inline]
```

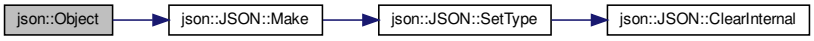
Definition at line 432 of file json.hpp.

References json::JSON::Make(), and json::JSON::Object.

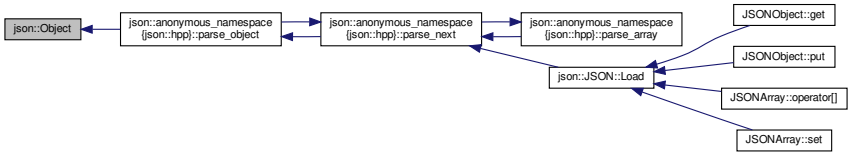
Referenced by json::anonymous_namespace{json.hpp}::parse_object().

```
432                       {
433     return std::move( JSON::Make( JSON::Class::Object ) );
434 }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.1.1.4 operator<<()

```
std::ostream& json::operator<< (
            std::ostream & os,
            const JSON & json )  [inline]
```

Definition at line 436 of file json.hpp.

```
436                                                                 {
437     os << json.dump();
438     return os;
439 }
```

## 7.2 json::anonymous_namespace{json.hpp} Namespace Reference

### Functions

- string json_escape (const string &str)
- JSON parse_next (const string &, size_t &)
- void consume_ws (const string &str, size_t &offset)
- JSON parse_object (const string &str, size_t &offset)
- JSON parse_array (const string &str, size_t &offset)
- JSON parse_string (const string &str, size_t &offset)
- JSON parse_number (const string &str, size_t &offset)
- JSON parse_bool (const string &str, size_t &offset)
- JSON parse_null (const string &str, size_t &offset)

### 7.2.1 Function Documentation

#### 7.2.1.1 consume_ws()

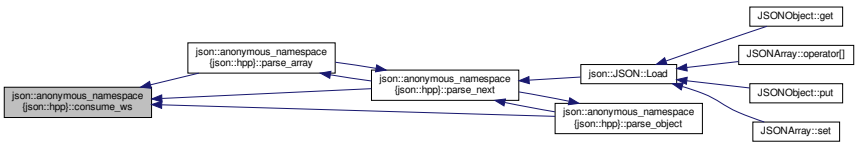
```
void json::anonymous_namespace{json.hpp}::consume_ws (
            const string & str,
            size_t & offset )
```

Definition at line 444 of file json.hpp.

Referenced by parse_array(), parse_next(), and parse_object().

```
444                                                     {
445         while( isspace( str[offset] ) ) ++offset;
446     }
```

Here is the caller graph for this function:



#### 7.2.1.2 json_escape()

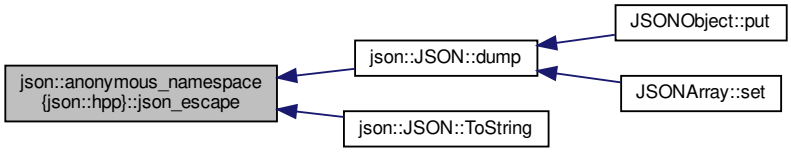
```
string json::anonymous_namespace{json.hpp}::json_escape (
            const string & str )
```

Definition at line 28 of file json.hpp.

Referenced by json::JSON::dump(), and json::JSON::ToString().

```
28                                          {
29         string output;
30         for( unsigned i = 0; i < str.length(); ++i )
31             switch( str[i] ) {
32                 case '\"': output += "\\\""; break;
33                 case '\\': output += "\\\\"; break;
34                 case '\b': output += "\\b";  break;
35                 case '\f': output += "\\f";  break;
36                 case '\n': output += "\\n";  break;
37                 case '\r': output += "\\r";  break;
38                 case '\t': output += "\\t";  break;
39                 default  : output += str[i]; break;
40             }
41         return std::move( output );
42     }
```

Here is the caller graph for this function:




#### 7.2.1.3 parse_array()

```
JSON json::anonymous_namespace{json.hpp}::parse_array (
            const string & str,
            size_t & offset )
```

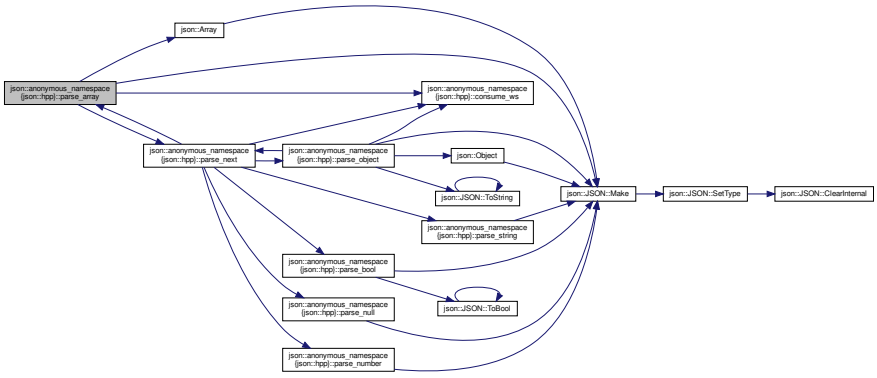
Definition at line 484 of file json.hpp.

References json::JSON::Array, json::Array(), consume_ws(), json::JSON::Make(), and parse_next().

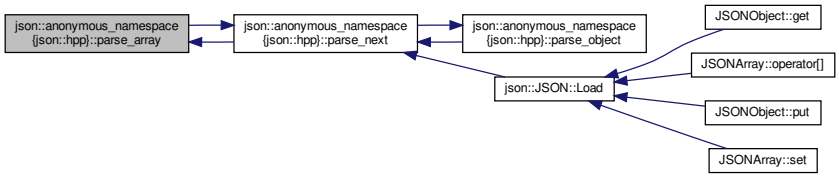
Referenced by parse_next().

```
484                                                 {
485         JSON Array = JSON::Make( JSON::Class::Array );
486         unsigned index = 0;
487 
488         ++offset;
489         consume_ws( str, offset );
490         if( str[offset] == ']' ) {
491             ++offset; return std::move( Array );
492         }
493 
494         while( true ) {
495             Array[index++] = parse_next( str, offset );
496             consume_ws( str, offset );
497 
498             if( str[offset] == ',' ) {
499                 ++offset; continue;
500             }
501             else if( str[offset] == ']' ) {
502                 ++offset; break;
503             }
504             else {
505                 std::cerr << "ERROR: Array: Expected ',' or ']', found '" << str[offset] << "'\n";
506                 return std::move( JSON::Make( JSON::Class::Array ) );
507             }
508         }
509 
510         return std::move( Array );
511     }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.2.1.4 parse_bool()

```
JSON json::anonymous_namespace{json.hpp}::parse_bool (
            const string & str,
            size_t & offset )
```

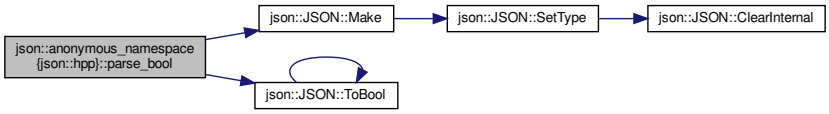
Definition at line 601 of file json.hpp.

References json::JSON::Make(), json::JSON::Null, and json::JSON::ToBool().

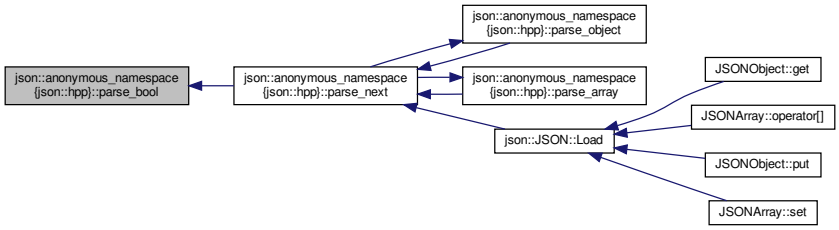
Referenced by parse_next().

```
601                                                     {
602         JSON Bool;
603         if( str.substr( offset, 4 ) == "true" )
604             Bool = true;
605         else if( str.substr( offset, 5 ) == "false" )
606             Bool = false;
607         else {
608             std::cerr << "ERROR: Bool: Expected 'true' or 'false', found '" << str.substr( offset, 5 ) << "
      '\n";
609             return std::move( JSON::Make( JSON::Class::Null ) );
610         }
611         offset += (Bool.ToBool() ? 4 : 5);
612         return std::move( Bool );
613     }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.2.1.5 parse_next()

```
JSON json::anonymous_namespace{json.hpp}::parse_next (
            const string & str,
            size_t & offset )
```

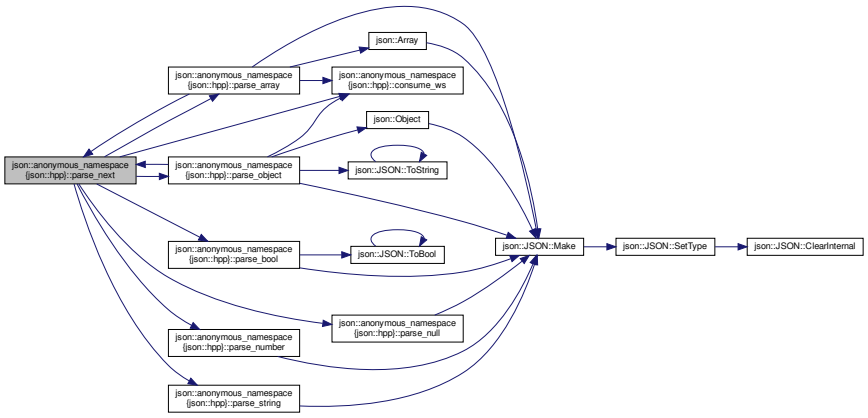
Definition at line 625 of file json.hpp.

References consume_ws(), parse_array(), parse_bool(), parse_null(), parse_number(), parse_object(), and parse↩
_string().

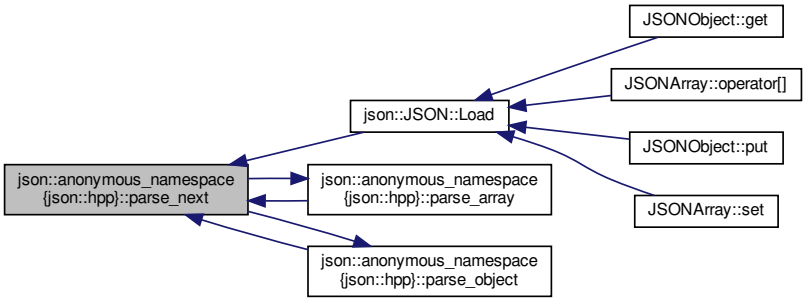
Referenced by json::JSON::Load(), parse_array(), and parse_object().

```
625                                                 {
626         char value;
627         consume_ws( str, offset );
628         value = str[offset];
629         switch( value ) {
630             case '[' : return std::move( parse_array( str, offset ) );
631             case '{' : return std::move( parse_object( str, offset ) );
632             case '\"': return std::move( parse_string( str, offset ) );
633             case 't' :
634             case 'f' : return std::move( parse_bool( str, offset ) );
635             case 'n' : return std::move( parse_null( str, offset ) );
636             default  : if( ( value <= '9' && value >= '0' ) || value == '-' )
637                             return std::move( parse_number( str, offset ) );
638         }
639         std::cerr << "ERROR: Parse: Unknown starting character '" << value << "'\n";
640         return JSON();
641     }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.2.1.6 parse_null()

```
JSON json::anonymous_namespace{json.hpp}::parse_null (
            const string & str,
            size_t & offset )
```

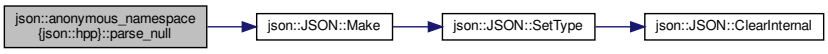
Definition at line 615 of file json.hpp.

References json::JSON::Make(), and json::JSON::Null.

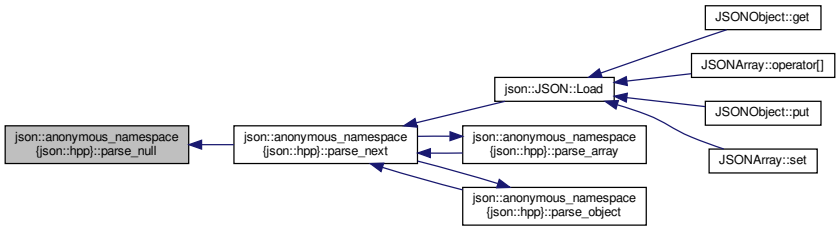
Referenced by parse_next().

```
615                                                     {
616         JSON Null;
617         if( str.substr( offset, 4 ) != "null" ) {
618             std::cerr << "ERROR: Null: Expected 'null', found '" << str.substr( offset, 4 ) << "'\n";
619             return std::move( JSON::Make( JSON::Class::Null ) );
620         }
621         offset += 4;
622         return std::move( Null );
623     }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.2.1.7 parse_number()

```
JSON json::anonymous_namespace{json.hpp}::parse_number (
            const string & str,
            size_t & offset )
```

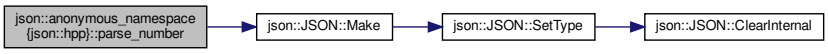
Definition at line 551 of file json.hpp.

References json::JSON::Make(), and json::JSON::Null.

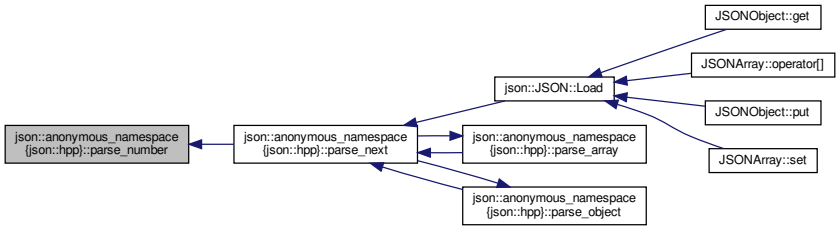
Referenced by parse_next().

```
551                                                     {
552         JSON Number;
553         string val, exp_str;
554         char c;
555         bool isDouble = false;
556         long exp = 0;
557         while( true ) {
558             c = str[offset++];
559             if( (c == '-') || (c >= '0' && c <= '9') )
560                 val += c;
561             else if( c == '.' ) {
562                 val += c;
563                 isDouble = true;
564             }
565             else
566                 break;
567         }
568         if( c == 'E' || c == 'e' ) {
569             c = str[ offset++ ];
570             if( c == '-' ){ ++offset; exp_str += '-';}
571             while( true ) {
572                 c = str[ offset++ ];
573                 if( c >= '0' && c <= '9' )
574                     exp_str += c;
575                 else if( !isspace( c ) && c != ',' && c != ']' && c != '}' ) {
576                     std::cerr << "ERROR: Number: Expected a number for exponent, found '" << c << "'\n";
577                     return std::move( JSON::Make( JSON::Class::Null ) );
578                 }
579                 else
580                     break;
581             }
582             exp = std::stol( exp_str );
583         }
584         else if( !isspace( c ) && c != ',' && c != ']' && c != '}' ) {
585             std::cerr << "ERROR: Number: unexpected character '" << c << "'\n";
586             return std::move( JSON::Make( JSON::Class::Null ) );
587         }
588         --offset;
589 
590         if( isDouble )
591             Number = std::stod( val ) * std::pow( 10, exp );
592         else {
593             if( !exp_str.empty() )
594                 Number = std::stol( val ) * std::pow( 10, exp );
595             else
596                 Number = std::stol( val );
597         }
598         return std::move( Number );
599     }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 7.2.1.8 parse_object()

```
JSON json::anonymous_namespace{json.hpp}::parse_object (
            const string & str,
            size_t & offset )
```

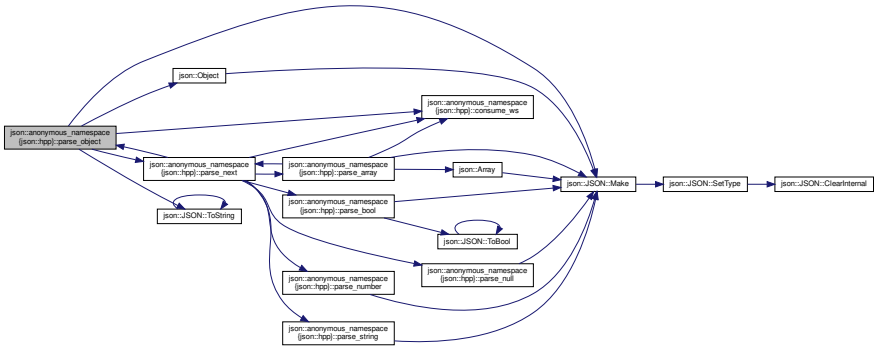
Definition at line 448 of file json.hpp.

References consume_ws(), json::JSON::Make(), json::JSON::Object, json::Object(), parse_next(), and json::JSON::ToString().

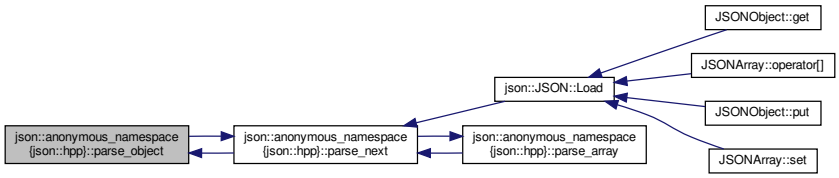
Referenced by parse_next().

```
448                                                     {
449         JSON Object = JSON::Make( JSON::Class::Object );
450
451         ++offset;
452         consume_ws( str, offset );
453         if( str[offset] == '}' ) {
454             ++offset; return std::move( Object );
455         }
456
457         while( true ) {
458             JSON Key = parse_next( str, offset );
459             consume_ws( str, offset );
460             if( str[offset] != ':' ) {
461                 std::cerr << "Error: Object: Expected colon, found '" << str[offset] << "'\n";
462                 break;
463             }
464             consume_ws( str, ++offset );
465             JSON Value = parse_next( str, offset );
466             Object[Key.ToString()] = Value;
467
468             consume_ws( str, offset );
469             if( str[offset] == ',' ) {
470                 ++offset; continue;
471             }
472             else if( str[offset] == '}' ) {
473                 ++offset; break;
474             }
475             else {
476                 std::cerr << "ERROR: Object: Expected comma, found '" << str[offset] << "'\n";
477                 break;
478             }
479         }
480
481         return std::move( Object );
482     }
```

Here is the call graph for this function:



Here is the caller graph for this function:




#### 7.2.1.9 parse_string()

```
JSON json::anonymous_namespace{json.hpp}::parse_string (
            const string & str,
            size_t & offset )
```

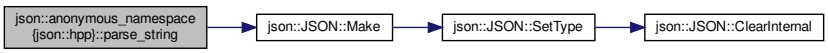
Definition at line 513 of file json.hpp.

References json::JSON::Make(), and json::JSON::String.

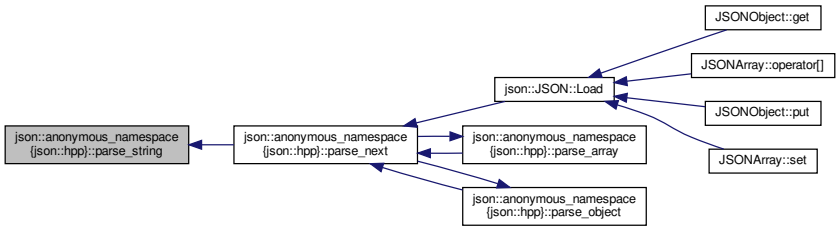
Referenced by parse_next().

```
513                                                         {
514         JSON String;
515         string val;
516         for( char c = str[++offset]; c != '\"' ; c = str[++offset] ) {
517             if( c == '\\' ) {
518                 switch( str[ ++offset ] ) {
519                 case '\"': val += '\"'; break;
520                 case '\\': val += '\\'; break;
521                 case '/' : val += '/' ; break;
522                 case 'b' : val += '\b'; break;
523                 case 'f' : val += '\f'; break;
524                 case 'n' : val += '\n'; break;
525                 case 'r' : val += '\r'; break;
526                 case 't' : val += '\t'; break;
527                 case 'u' : {
528                     val += "\\u" ;
529                     for( unsigned i = 1; i <= 4; ++i ) {
530                         c = str[offset+i];
531                         if( (c >= '0' && c <= '9') || (c >= 'a' && c <= 'f') || (c >= 'A' && c <= 'F') )
532                             val += c;
533                         else {
534                             std::cerr << "ERROR: String: Expected hex character in unicode escape, found '"
      << c << "'\n";
535                             return std::move( JSON::Make( JSON::Class::String ) );
536                         }
537                     }
538                     offset += 4;
539                 } break;
540                 default  : val += '\\'; break;
541                 }
542             }
543             else
544                 val += c;
545         }
546         ++offset;
547         String = val;
548         return std::move( String );
549     }
```

Here is the call graph for this function:




Here is the caller graph for this function:

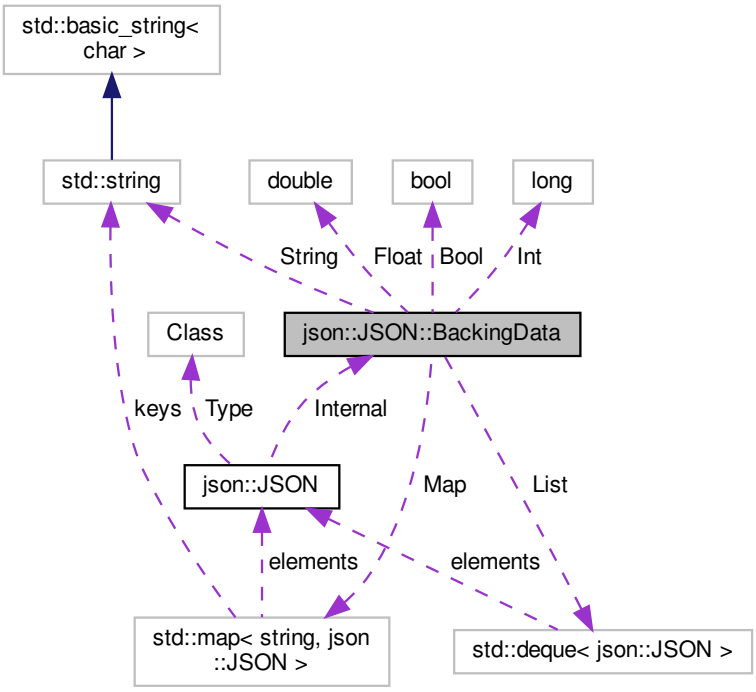


# Chapter 8

# Data Structure Documentation

## 8.1 json::JSON::BackingData Union Reference

Collaboration diagram for json::JSON::BackingData:




### Public Member Functions

- BackingData (double d)
- BackingData (long l)
- BackingData (bool b)
- BackingData (string s)
- BackingData ()

### Data Fields

- deque< JSON > * List
- map< string, JSON > * Map
- string * String
- double Float
- long Int
- bool Bool

## 8.1.1 Detailed Description

Definition at line 47 of file json.hpp.

## 8.1.2 Constructor & Destructor Documentation

### 8.1.2.1 BackingData() [1/5]

```
json::JSON::BackingData::BackingData (
            double d )  [inline]
```

Definition at line 48 of file json.hpp.

```
48 : Float( d ){}
```

### 8.1.2.2 BackingData() [2/5]

```
json::JSON::BackingData::BackingData (
            long l )  [inline]
```

Definition at line 49 of file json.hpp.

```
49 : Int( l ){}
```

### 8.1.2.3 BackingData() [3/5]

```
json::JSON::BackingData::BackingData (
            bool b )  [inline]
```

Definition at line 50 of file json.hpp.

```
50 : Bool( b ){}
```

#### 8.1.2.4 BackingData() [4/5]

```
json::JSON::BackingData::BackingData (
            string s )  [inline]
```

Definition at line 51 of file json.hpp.

```
51 : String( new string( s ) ){}
```

#### 8.1.2.5 BackingData() [5/5]

```
json::JSON::BackingData::BackingData ( )  [inline]
```

Definition at line 52 of file json.hpp.

```
52 : Int( 0 ){}
```

### 8.1.3 Field Documentation

#### 8.1.3.1 Bool

```
bool json::JSON::BackingData::Bool
```

Definition at line 59 of file json.hpp.

Referenced by json::JSON::dump(), json::JSON::operator=(), json::JSON::SetType(), and json::JSON::ToBool().

#### 8.1.3.2 Float

```
double json::JSON::BackingData::Float
```

Definition at line 57 of file json.hpp.

Referenced by json::JSON::dump(), json::JSON::operator=(), json::JSON::SetType(), and json::JSON::ToFloat().

#### 8.1.3.3 Int

```
long json::JSON::BackingData::Int
```

Definition at line 58 of file json.hpp.

Referenced by json::JSON::dump(), json::JSON::operator=(), json::JSON::SetType(), and json::JSON::ToInt().

#### 8.1.3.4 List

```
deque<JSON>* json::JSON::BackingData::List
```

Definition at line 54 of file json.hpp.

Referenced by json::JSON::append(), json::JSON::ArrayRange(), json::JSON::at(), json::JSON::ClearInternal(), json::JSON::dump(), json::JSON::JSON(), json::JSON::length(), json::JSON::operator=(), json::JSON::operator[ ](), json::JSON::SetType(), json::JSON::size(), and json::JSON::∼JSON().

#### 8.1.3.5 Map

```
map<string,JSON>* json::JSON::BackingData::Map
```

Definition at line 55 of file json.hpp.

Referenced by json::JSON::at(), json::JSON::ClearInternal(), json::JSON::dump(), json::JSON::hasKey(), json::↩ JSON::JSON(), json::JSON::ObjectRange(), json::JSON::operator=(), json::JSON::operator[ ](), json::JSON::Set↩ Type(), json::JSON::size(), and json::JSON::∼JSON().

#### 8.1.3.6 String

```
string* json::JSON::BackingData::String
```

Definition at line 56 of file json.hpp.

Referenced by json::JSON::ClearInternal(), json::JSON::dump(), json::JSON::JSON(), json::JSON::operator=(), json::JSON::SetType(), json::JSON::ToString(), and json::JSON::∼JSON().

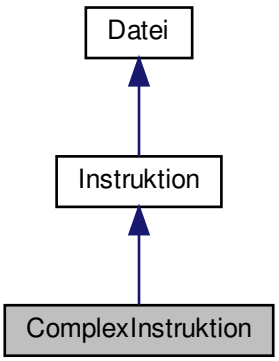
The documentation for this union was generated from the following file:

- lib/simplejson/json.hpp

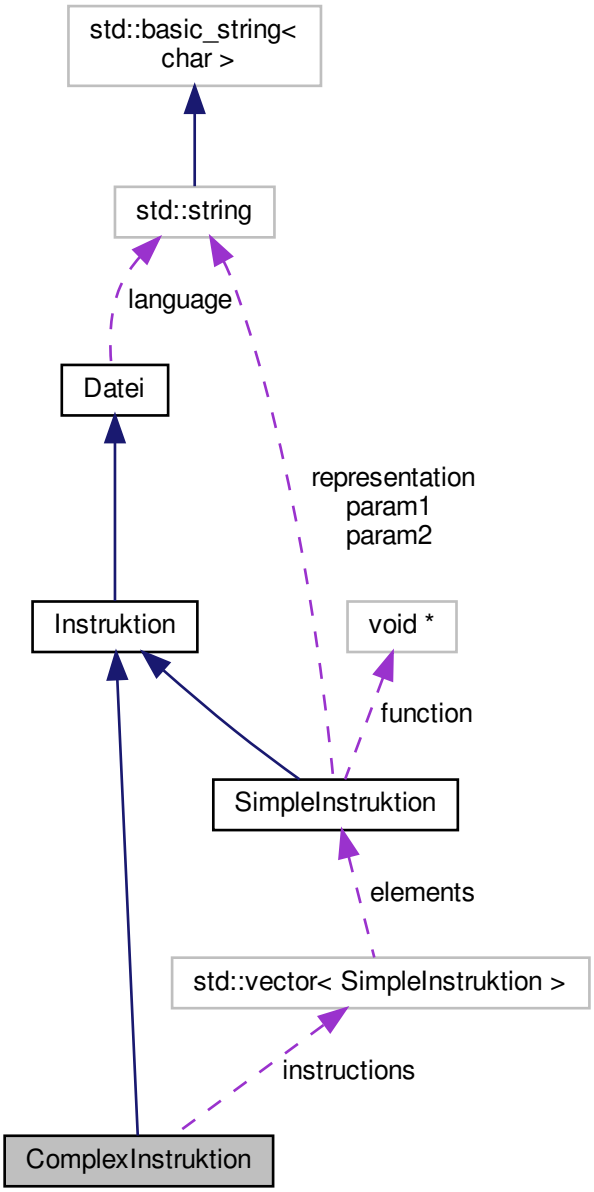
## 8.2 ComplexInstruktion Class Reference

`#include <instruction.hpp>`

Inheritance diagram for ComplexInstruktion:



Collaboration diagram for ComplexInstruktion:




## Public Member Functions

- ComplexInstruktion (std::string)
- ComplexInstruktion (std::string, SimpleInstruktion &)
- std::string print ()
- void run (VirtualMachine &)
- void add (SimpleInstruktion &)

### Private Attributes

- std::vector< SimpleInstruktion > instructions

### Additional Inherited Members

## 8.2.1 Detailed Description

Klasse für mehrerern Instruktionen. Funktioniert wie eine klügere Array (aber komplexer, auch).

Definition at line 42 of file instruction.hpp.

## 8.2.2 Constructor & Destructor Documentation

### 8.2.2.1 ComplexInstruktion() [1/2]

```
ComplexInstruktion::ComplexInstruktion (
            std::string lang )
```

Initialisiert ComplexInstruktion nur mit der Sprache.

Definition at line 35 of file instruction.cpp.

```
35                                                  : Instruktion(lang)
36 {
37 }
```

### 8.2.2.2 ComplexInstruktion() [2/2]

```
ComplexInstruktion::ComplexInstruktion (
            std::string lang,
            SimpleInstruktion & begin )
```

Initialisiert ComplexInstruktion sowohl mit der Sprache als auch mit einem SimpleInstruktion.

Definition at line 43 of file instruction.cpp.

References instructions.

```
43                                                                  :
     Instruktion(lang)
44 {
45     instructions.push_back(begin);
46 }
```

### 8.2.3 Member Function Documentation

#### 8.2.3.1 add()

```
void ComplexInstruktion::add (
            SimpleInstruktion & si )
```

Eine neue SimpleInstruktion einfügen.

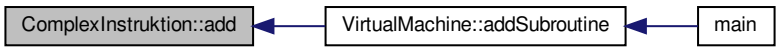
Definition at line 77 of file instruction.cpp.

References instructions.

Referenced by VirtualMachine::addSubroutine().

```
78 {
79     instructions.push_back(si);
80 }
```

Here is the caller graph for this function:




#### 8.2.3.2 print()

```
std::string ComplexInstruktion::print ( )  [virtual]
```

Gibt die String-Representation des ganzen Instruktionenliste zurück.

Implements Instruktion.

Definition at line 52 of file instruction.cpp.

References instructions.

```
53 {
54     std::string ret = "";
55     for(SimpleInstruktion si : instructions)
56     {
57         ret+=si.print()+='\n';
58     }
59     return ret;
60 }
```

#### 8.2.3.3 run()

```
void ComplexInstruktion::run (
            VirtualMachine & vm )  [virtual]
```

Ausführt jede Instruktion, denen sie enthielt.

Implements Instruktion.

Definition at line 65 of file instruction.cpp.

References instructions.

```
66 {
67     for(SimpleInstruktion si : instructions)
68     {
69         si.run(vm);
70     }
71 }
```

### 8.2.4 Field Documentation

#### 8.2.4.1 instructions

```
std::vector<SimpleInstruktion> ComplexInstruktion::instructions  [private]
```

Definition at line 45 of file instruction.hpp.

Referenced by add(), ComplexInstruktion(), print(), and run().

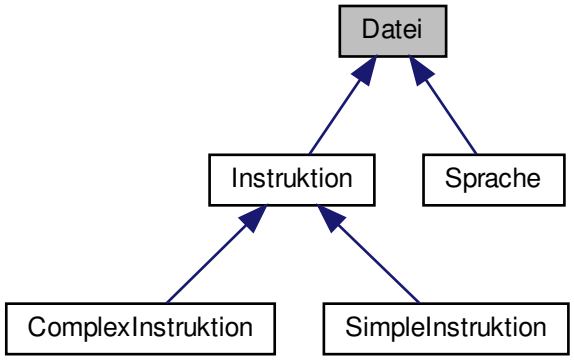
The documentation for this class was generated from the following files:

- src/datei/instruction/instruction.hpp
- src/datei/instruction/instruction.cpp

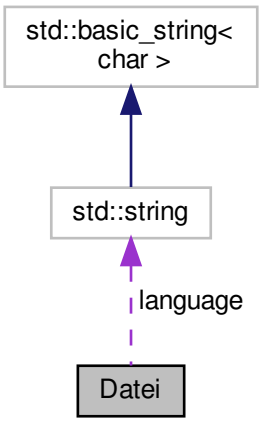
## 8.3 Datei Class Reference

`#include <datei.hpp>`

Inheritance diagram for Datei:




Collaboration diagram for Datei:




### Public Member Functions

- Datei (std::string)
- virtual std::string print ()=0
- std::string getLang ()

### Protected Attributes

- std::string language

### Friends

- std::ostream & operator<< (std::ostream &, Datei &)

## 8.3.1 Detailed Description

Klasse für Dateien.

Definition at line 10 of file datei.hpp.

## 8.3.2 Constructor & Destructor Documentation

### 8.3.2.1 Datei()

```
Datei::Datei (
            std::string lang )
```

Konstruktor für Dateien, stellt die Sprache ein.

Definition at line 7 of file datei.cpp.

```
7 : language(lang) {}
```

## 8.3.3 Member Function Documentation

#### 8.3.3.1 getLang()

```
std::string Datei::getLang ( )
```

Getterfunktion für die Sprache.

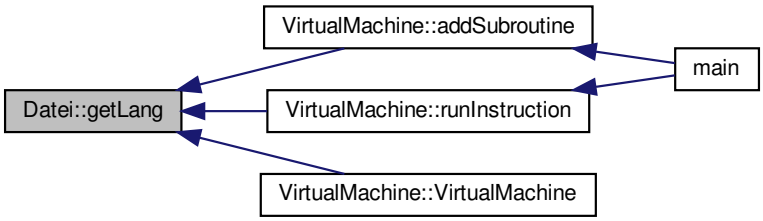
Definition at line 22 of file datei.cpp.

References language.

Referenced by VirtualMachine::addSubroutine(), VirtualMachine::runInstruction(), and VirtualMachine::Virtual↩ Machine().

```
22 { return language;}
```

Here is the caller graph for this function:




#### 8.3.3.2 print()

```
virtual std::string Datei::print ( )  [pure virtual]
```

Implemented in ComplexInstruktion, SimpleInstruktion, Sprache, and Instruktion.

Referenced by operator<<().

Here is the caller graph for this function:

### 8.3.4 Friends And Related Function Documentation

#### 8.3.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            Datei & d )  [friend]
```

Auschreiben.

Definition at line 13 of file datei.cpp.

```
14 {
15     os<<d.print();
16     return os;
17 }
```

### 8.3.5 Field Documentation

#### 8.3.5.1 language

```
std::string Datei::language  [protected]
```

Definition at line 12 of file datei.hpp.

Referenced by getLang().

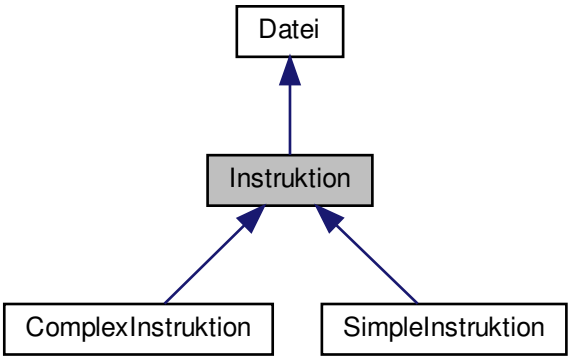
The documentation for this class was generated from the following files:

- src/datei/datei.hpp
- src/datei/datei.cpp

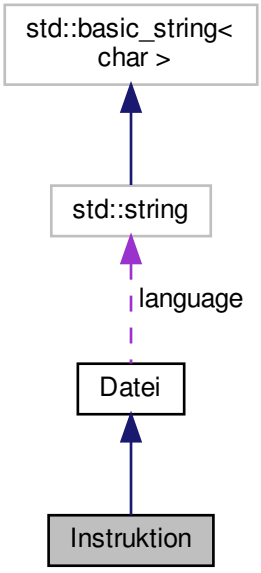
## 8.4 Instruktion Class Reference

`#include <instruction.hpp>`

Inheritance diagram for Instruktion:




Collaboration diagram for Instruktion:




### Public Member Functions

- Instruktion (std::string str)
- virtual std::string print ()=0
- virtual void run (VirtualMachine &)=0

### Additional Inherited Members

## 8.4.1 Detailed Description

Den Basisklasse für die Instruktionentypen.

Definition at line 13 of file instruction.hpp.

## 8.4.2 Constructor & Destructor Documentation

### 8.4.2.1 Instruktion()

```
Instruktion::Instruktion (
            std::string str )  [inline]
```

Definition at line 16 of file instruction.hpp.

```
16 : Datei(str) {}    /* Konstruktor der Instruktionklasse */
```

## 8.4.3 Member Function Documentation

### 8.4.3.1 print()

```
virtual std::string Instruktion::print ( )  [pure virtual]
```

Implements Datei.

Implemented in ComplexInstruktion, and SimpleInstruktion.

### 8.4.3.2 run()

```
virtual void Instruktion::run (
            VirtualMachine &  )  [pure virtual]
```

Implemented in ComplexInstruktion, and SimpleInstruktion.

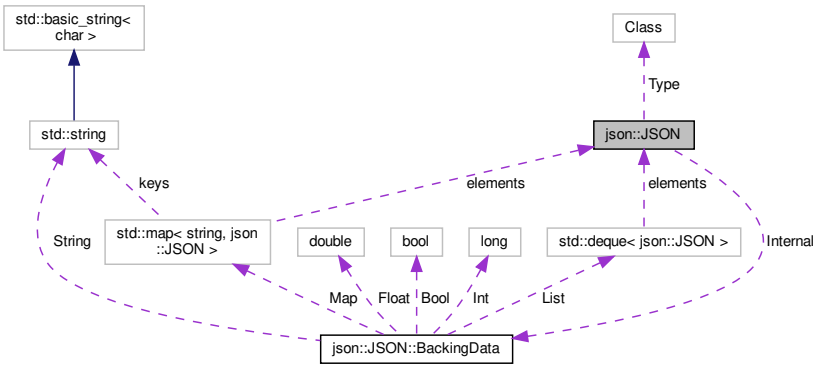
The documentation for this class was generated from the following file:

- src/datei/instruction/instruction.hpp

## 8.5 json::JSON Class Reference

`#include <json.hpp>`

Collaboration diagram for json::JSON:




### Data Structures

- union BackingData
- class JSONConstWrapper
- class JSONWrapper

### Public Types

- enum Class {
  Class::Null, Class::Object, Class::Array, Class::String,
  Class::Floating, Class::Integral, Class::Boolean }

### Public Member Functions

- JSON ()
- JSON (initializer_list< JSON > list)
- JSON (JSON &&other)
- JSON & operator= (JSON &&other)
- JSON (const JSON &other)
- JSON & operator= (const JSON &other)
- ~JSON ()
- template<typename T >
  JSON (T b, typename enable_if< is_same< T, bool >::value >::type *=0)
- template<typename T >
  JSON (T i, typename enable_if< is_integral< T >::value &&!is_same< T, bool >::value >::type *=0)
- template<typename T >
  JSON (T f, typename enable_if< is_floating_point< T >::value >::type *=0)
- template<typename T >
  JSON (T s, typename enable_if< is_convertible< T, string >::value >::type *=0)
- JSON (std::nullptr_t)

- template<typename T >
  void append (T arg)
- template<typename T , typename... U>
  void append (T arg, U... args)
- template<typename T >
  enable_if< is_same< T, bool >::value, JSON & >::type operator= (T b)
- template<typename T >
  enable_if< is_integral< T >::value &&!is_same< T, bool >::value, JSON & >::type operator= (T i)
- template<typename T >
  enable_if< is_floating_point< T >::value, JSON & >::type operator= (T f)
- template<typename T >
  enable_if< is_convertible< T, string >::value, JSON & >::type operator= (T s)
- JSON & operator[ ] (const string &key)
- JSON & operator[ ] (unsigned index)
- JSON & at (const string &key)
- const JSON & at (const string &key) const
- JSON & at (unsigned index)
- const JSON & at (unsigned index) const
- int length () const
- bool hasKey (const string &key) const
- int size () const
- Class JSONType () const
- bool IsNull () const

  *Functions for getting primitives from the JSON object.*
- string ToString () const
- string ToString (bool &ok) const
- double ToFloat () const
- double ToFloat (bool &ok) const
- long ToInt () const
- long ToInt (bool &ok) const
- bool ToBool () const
- bool ToBool (bool &ok) const
- JSONWrapper< map< string, JSON > > ObjectRange ()
- JSONWrapper< deque< JSON > > ArrayRange ()
- JSONConstWrapper< map< string, JSON > > ObjectRange () const
- JSONConstWrapper< deque< JSON > > ArrayRange () const
- string dump (int depth=1, string tab=" ") const

### Static Public Member Functions

- static JSON Make (Class type)
- static JSON Load (const string &)

### Private Member Functions

- void SetType (Class type)
- void ClearInternal ()

### Private Attributes

- union json::JSON::BackingData Internal
- Class Type = Class::Null

### Friends

- std::ostream & operator<< (std::ostream &, const JSON &)

## 8.5.1 Detailed Description

Definition at line 45 of file json.hpp.

## 8.5.2 Member Enumeration Documentation

### 8.5.2.1 Class

```
enum json::JSON::Class  [strong]
```

**Enumerator**

| | |
|---|---|
| Null | |
| Object | |
| Array | |
| String | |
| Floating | |
| Integral | |
| Boolean | |

Definition at line 63 of file json.hpp.

```
63                          {
64             Null,
65             Object,
66             Array,
67             String,
68             Floating,
69             Integral,
70             Boolean
71         };
```

## 8.5.3 Constructor & Destructor Documentation

### 8.5.3.1 JSON() [1/9]

```
json::JSON::JSON ( )  [inline]
```

Definition at line 99 of file json.hpp.

```
99 : Internal(), Type( Class::Null ){}
```

#### 8.5.3.2 JSON() [2/9]

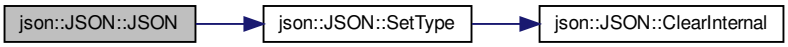
```
json::JSON::JSON (
            initializer_list< JSON > list )  [inline]
```

Definition at line 101 of file json.hpp.

References Object, and SetType().

```
102             : JSON()
103         {
104             SetType( Class::Object );
105             for( auto i = list.begin(), e = list.end(); i != e; ++i, ++i )
106                 operator[]( i->ToString() ) = *std::next( i );
107         }
```

Here is the call graph for this function:




#### 8.5.3.3 JSON() [3/9]

```
json::JSON::JSON (
            JSON && other )  [inline]
```

Definition at line 109 of file json.hpp.

References Null.

```
110             : Internal( other.Internal )
111             , Type( other.Type )
112         { other.Type = Class::Null; other.Internal.Map = nullptr; }
```

#### 8.5.3.4 JSON() [4/9]

```
json::JSON::JSON (
            const JSON & other )  [inline]
```

Definition at line 123 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, json::JSON::BackingData::Map, Object, json::JSON::↩ BackingData::String, String, and Type.

```
123                                 {
124             switch( other.Type ) {
125             case Class::Object:
126                 Internal.Map =
127                     new map<string,JSON>( other.Internal.Map->begin(),
128                                           other.Internal.Map->end() );
129                 break;
130             case Class::Array:
131                 Internal.List =
132                     new deque<JSON>( other.Internal.List->begin(),
133                                      other.Internal.List->end() );
134                 break;
135             case Class::String:
136                 Internal.String =
137                     new string( *other.Internal.String );
138                 break;
139             default:
140                 Internal = other.Internal;
141             }
142             Type = other.Type;
143         }
```

#### 8.5.3.5 ~JSON()

```
json::JSON::~JSON ( )  [inline]
```

Definition at line 169 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, json::JSON::BackingData::Map, Object, json::JSON::↩ BackingData::String, String, and Type.

```
169             {
170             switch( Type ) {
171             case Class::Array:
172                 delete Internal.List;
173                 break;
174             case Class::Object:
175                 delete Internal.Map;
176                 break;
177             case Class::String:
178                 delete Internal.String;
179                 break;
180             default:;
181             }
182         }
```

#### 8.5.3.6 JSON() [5/9]

```
template<typename T >
json::JSON::JSON (
            T b,
            typename enable_if< is_same< T, bool >::value >::type *  = 0 )  [inline]
```

Definition at line 185 of file json.hpp.

```
185 : Internal( b ), Type( Class::Boolean ){}
```

#### 8.5.3.7 JSON() [6/9]

```
template<typename T >
json::JSON::JSON (
            T i,
            typename enable_if< is_integral< T >::value &&!is_same< T, bool >::value >↩
::type *  = 0 )  [inline]
```

Definition at line 188 of file json.hpp.

```
188 : Internal( (long)i ), Type( Class::Integral ){}
```

#### 8.5.3.8 JSON() [7/9]

```
template<typename T >
json::JSON::JSON (
            T f,
            typename enable_if< is_floating_point< T >::value >::type *  = 0 )  [inline]
```

Definition at line 191 of file json.hpp.

```
191 : Internal( (double)f ), Type( Class::Floating ){}
```

#### 8.5.3.9 JSON() [8/9]

```
template<typename T >
json::JSON::JSON (
            T s,
            typename enable_if< is_convertible< T, string >::value >::type *  = 0 )  [inline]
```

Definition at line 194 of file json.hpp.

```
194 : Internal( string( s ) ), Type( Class::String ){}
```

#### 8.5.3.10 JSON() [9/9]

```
json::JSON::JSON (
            std::nullptr_t  )  [inline]
```

Definition at line 196 of file json.hpp.

```
196 : Internal(), Type( Class::Null ){}
```

### 8.5.4 Member Function Documentation

#### 8.5.4.1 append() [1/2]

```
template<typename T >
void json::JSON::append (
            T arg )  [inline]
```

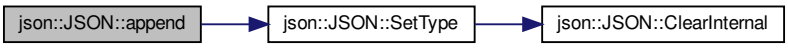
Definition at line 206 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, and SetType().

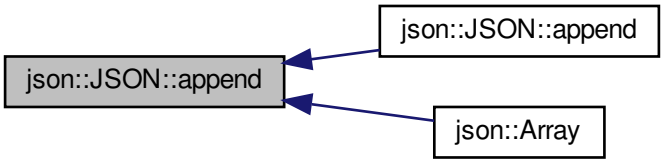
Referenced by append(), and json::Array().

```
206                             {
207             SetType( Class::Array ); Internal.List->emplace_back( arg );
208         }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 8.5.4.2 append() [2/2]

```
template<typename T , typename...  U>
void json::JSON::append (
            T arg,
            U...  args )  [inline]
```

Definition at line 211 of file json.hpp.

References append().

```
211                                         {
212             append( arg ); append( args... );
213         }
```

Here is the call graph for this function:


#### 8.5.4.3 ArrayRange() [1/2]

```
JSONWrapper<deque<JSON> > json::JSON::ArrayRange ( )  [inline]
```

Definition at line 318 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, and Type.

```
318                                             {
319             if( Type == Class::Array )
320                 return JSONWrapper<deque<JSON>>( Internal.List );
321             return JSONWrapper<deque<JSON>>( nullptr );
322         }
```

#### 8.5.4.4 ArrayRange() [2/2]

```
JSONConstWrapper<deque<JSON> > json::JSON::ArrayRange ( ) const  [inline]
```

Definition at line 331 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, and Type.

```
331                                                   {
332             if( Type == Class::Array )
333                 return JSONConstWrapper<deque<JSON>>( Internal.List );
334             return JSONConstWrapper<deque<JSON>>( nullptr );
335         }
```

#### 8.5.4.5 at() [1/4]

```
JSON& json::JSON::at (
            const string & key )  [inline]
```

Definition at line 245 of file json.hpp.

References operator[]().

```
245                                     {
246             return operator[]( key );
247         }
```

Here is the call graph for this function:


#### 8.5.4.6 at() [2/4]

```
const JSON& json::JSON::at (
            const string & key ) const  [inline]
```

Definition at line 249 of file json.hpp.

References Internal, and json::JSON::BackingData::Map.

```
249                                                 {
250             return Internal.Map->at( key );
251         }
```

#### 8.5.4.7 at() [3/4]

```
JSON& json::JSON::at (
            unsigned index )  [inline]
```

Definition at line 253 of file json.hpp.

References operator[]().

```
253                                     {
254             return operator[]( index );
255         }
```

Here is the call graph for this function:

#### 8.5.4.8 at() [4/4]

```
const JSON& json::JSON::at (
            unsigned index ) const  [inline]
```

Definition at line 257 of file json.hpp.

References Internal, and json::JSON::BackingData::List.

```
257                                             {
258             return Internal.List->at( index );
259         }
```

#### 8.5.4.9 ClearInternal()

```
void json::JSON::ClearInternal ( )  [inline], [private]
```

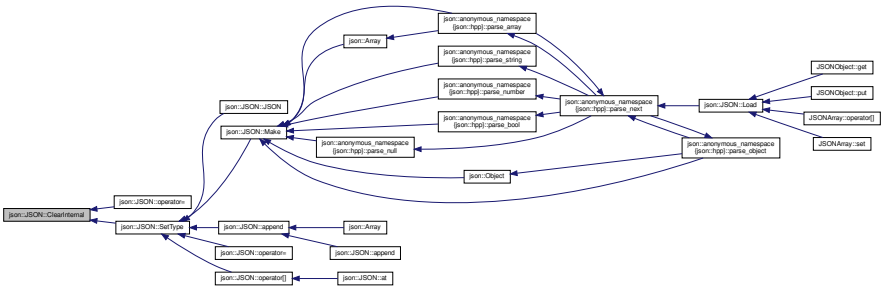
Definition at line 407 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, json::JSON::BackingData::Map, Object, json::JSON::↩ BackingData::String, String, and Type.

Referenced by operator=(), and SetType().

```
407                             {
408         switch( Type ) {
409           case Class::Object: delete Internal.Map;    break;
410           case Class::Array:  delete Internal.List;   break;
411           case Class::String: delete Internal.String; break;
412           default:;
413         }
414       }
```

Here is the caller graph for this function:



#### 8.5.4.10 dump()

```
string json::JSON::dump (
            int depth = 1,
            string tab = "  " ) const  [inline]
```

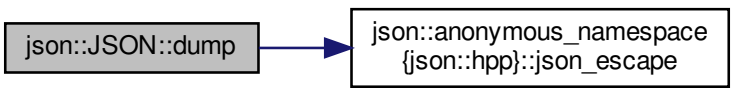
Definition at line 337 of file json.hpp.

References Array, json::JSON::BackingData::Bool, Boolean, json::JSON::BackingData::Float, Floating, json::JSON::BackingData::Int, Integral, Internal, json::anonymous_namespace{json.hpp}::json_escape(), json::JSON::BackingData::List, json::JSON::BackingData::Map, Null, Object, json::JSON::BackingData::String, String, and Type.

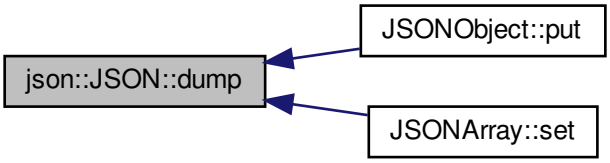
Referenced by JSONObject::put(), and JSONArray::set().

```
337                                                         {
338             string pad = "";
339             for( int i = 0; i < depth; ++i, pad += tab );
340 
341             switch( Type ) {
342                 case Class::Null:
343                     return "null";
344                 case Class::Object: {
345                     string s = "{\n";
346                     bool skip = true;
347                     for( auto &p : *Internal.Map ) {
348                         if( !skip ) s += ",\n";
349                         s += ( pad + "" + p.first + " : " + p.second.dump( depth + 1, tab ) );
350                         skip = false;
351                     }
352                     s += ( "\n" + pad.erase( 0, 2 ) + "}" ) ;
353                     return s;
354                 }
355                 case Class::Array: {
356                     string s = "[";
357                     bool skip = true;
358                     for( auto &p : *Internal.List ) {
359                         if( !skip ) s += ", ";
360                         s += p.dump( depth + 1, tab );
361                         skip = false;
362                     }
363                     s += "]";
364                     return s;
365                 }
366                 case Class::String:
367                     return "" + json_escape( *Internal.String ) + "";
368                 case Class::Floating:
369                     return std::to_string( Internal.Float );
370                 case Class::Integral:
371                     return std::to_string( Internal.Int );
372                 case Class::Boolean:
373                     return Internal.Bool ? "true" : "false";
374                 default:
375                     return "";
376             }
377             return "";
378         }
```

Here is the call graph for this function:



Here is the caller graph for this function:




#### 8.5.4.11 hasKey()

```
bool json::JSON::hasKey (
              const string & key ) const  [inline]
```

Definition at line 268 of file json.hpp.

References Internal, json::JSON::BackingData::Map, Object, and Type.

```
268                                                 {
269             if( Type == Class::Object )
270                 return Internal.Map->find( key ) != Internal.
    Map->end();
271             return false;
272         }
```

#### 8.5.4.12 IsNull()

```
bool json::JSON::IsNull ( ) const  [inline]
```

Functions for getting primitives from the JSON object.

Definition at line 286 of file json.hpp.

References Null, and Type.

```
286 { return Type == Class::Null; }
```

#### 8.5.4.13 JSONType()

```
Class json::JSON::JSONType ( ) const  [inline]
```

Definition at line 283 of file json.hpp.

References Type.

```
283 { return Type; }
```

#### 8.5.4.14 length()

```
int json::JSON::length ( ) const  [inline]
```

Definition at line 261 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, and Type.

```
261                             {
262             if( Type == Class::Array )
263                 return Internal.List->size();
264             else
265                 return -1;
266         }
```

#### 8.5.4.15 Load()

```
JSON JSON::Load (
             const string & str )  [inline], [static]
```

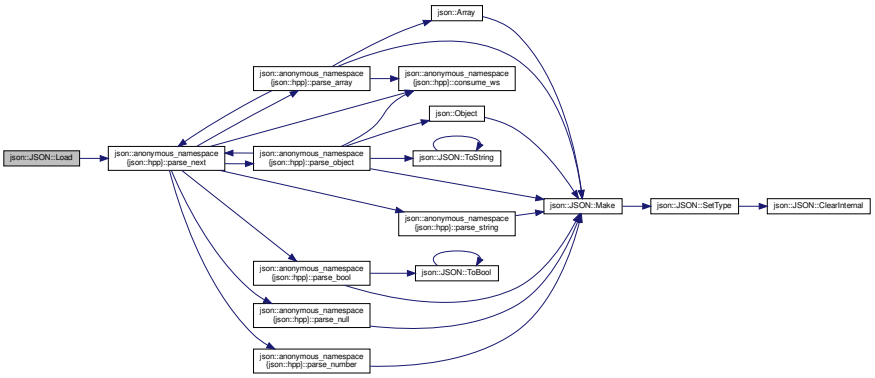
Definition at line 644 of file json.hpp.

References json::anonymous_namespace{json.hpp}::parse_next().

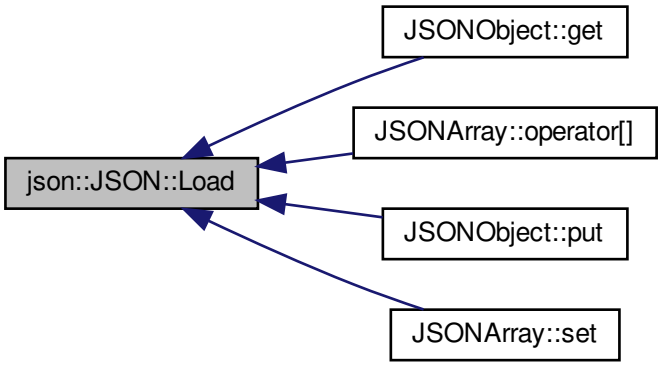
Referenced by JSONObject::get(), JSONArray::operator[](), JSONObject::put(), and JSONArray::set().

```
644                                         {
645     size_t offset = 0;
646     return std::move( parse_next( str, offset ) );
647 }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 8.5.4.16 Make()

```
static JSON json::JSON::Make (
            Class type )  [inline], [static]
```

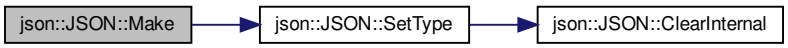
Definition at line 198 of file json.hpp.

References SetType().

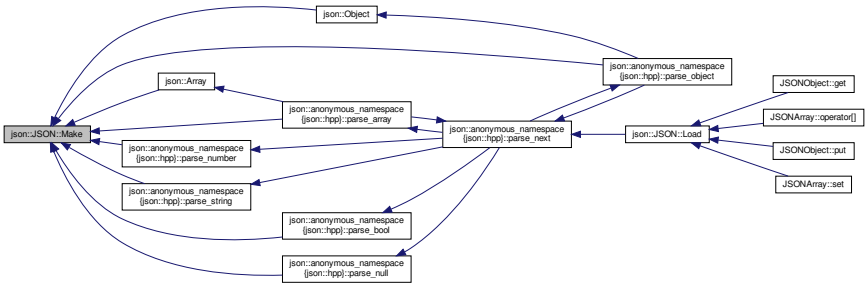
Referenced by json::Array(), json::Object(), json::anonymous_namespace{json.hpp}::parse_array(), json::anonymous_namespace{json.hpp}::parse_bool(), json::anonymous_namespace{json.hpp}::parse_null(), json::anonymous_namespace{json.hpp}::parse_number(), json::anonymous_namespace{json.hpp}::parse_object(), and json::anonymous_namespace{json.hpp}::parse_string().

```
198                                     {
199             JSON ret; ret.SetType( type );
200             return ret;
201         }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 8.5.4.17 ObjectRange() [1/2]

`JSONWrapper<map<string,JSON> > json::JSON::ObjectRange ( ) [inline]`

Definition at line 312 of file json.hpp.

References Internal, json::JSON::BackingData::Map, Object, and Type.

```
312                                                 {
313             if( Type == Class::Object )
314                 return JSONWrapper<map<string,JSON>>( Internal.Map );
315             return JSONWrapper<map<string,JSON>>( nullptr );
316         }
```

#### 8.5.4.18 ObjectRange() [2/2]

`JSONConstWrapper<map<string,JSON> > json::JSON::ObjectRange ( ) const [inline]`

Definition at line 324 of file json.hpp.

References Internal, json::JSON::BackingData::Map, Object, and Type.

```
324                                                       {
325             if( Type == Class::Object )
326                 return JSONConstWrapper<map<string,JSON>>( Internal.Map );
327             return JSONConstWrapper<map<string,JSON>>( nullptr );
328         }
```

#### 8.5.4.19 operator=() [1/6]

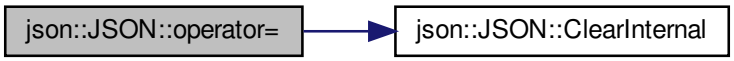
```
JSON& json::JSON::operator= (
            JSON && other )  [inline]
```

Definition at line 114 of file json.hpp.

References ClearInternal(), Internal, Null, and Type.

```
114                                     {
115             ClearInternal();
116             Internal = other.Internal;
117             Type = other.Type;
118             other.Internal.Map = nullptr;
119             other.Type = Class::Null;
120             return *this;
121         }
```

Here is the call graph for this function:




#### 8.5.4.20 operator=() [2/6]

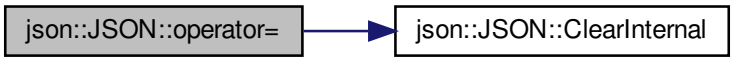
```
JSON& json::JSON::operator= (
            const JSON & other )  [inline]
```

Definition at line 145 of file json.hpp.

References Array, ClearInternal(), Internal, json::JSON::BackingData::List, json::JSON::BackingData::Map, Object, json::JSON::BackingData::String, String, and Type.

```
145                                           {
146             ClearInternal();
147             switch( other.Type ) {
148             case Class::Object:
149                 Internal.Map =
150                     new map<string,JSON>( other.Internal.Map->begin(),
151                                           other.Internal.Map->end() );
152                 break;
153             case Class::Array:
154                 Internal.List =
155                     new deque<JSON>( other.Internal.List->begin(),
156                                      other.Internal.List->end() );
157                 break;
158             case Class::String:
159                 Internal.String =
160                     new string( *other.Internal.String );
161                 break;
162             default:
163                 Internal = other.Internal;
164             }
165             Type = other.Type;
166             return *this;
167         }
```

Here is the call graph for this function:




#### 8.5.4.21 operator=() [3/6]

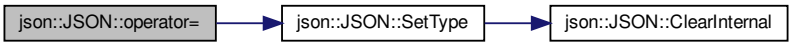
```
template<typename T >
enable_if<is_same<T,bool>::value, JSON&>::type json::JSON::operator= (
            T b )  [inline]
```

Definition at line 216 of file json.hpp.

References json::JSON::BackingData::Bool, Boolean, Internal, and SetType().

```
216                                                                          {
217                 SetType( Class::Boolean ); Internal.
      Bool = b; return *this;
218             }
```

Here is the call graph for this function:




#### 8.5.4.22 operator=() [4/6]

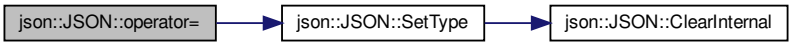
```
template<typename T >
enable_if<is_integral<T>::value && !is_same<T,bool>::value, JSON&>::type json::JSON::operator=
(
            T i )  [inline]
```

Definition at line 221 of file json.hpp.

References json::JSON::BackingData::Int, Integral, Internal, and SetType().

```
221                                                                                                                 {
222                 SetType( Class::Integral ); Internal.
      Int = i; return *this;
223             }
```

Here is the call graph for this function:




#### 8.5.4.23 operator=() [5/6]

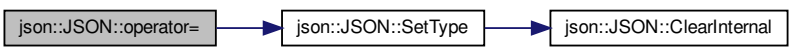
```
template<typename T >
enable_if<is_floating_point<T>::value, JSON&>::type json::JSON::operator= (
              T f )  [inline]
```

Definition at line 226 of file json.hpp.

References json::JSON::BackingData::Float, Floating, Internal, and SetType().

```
226                                                                                                                 {
227                 SetType( Class::Floating ); Internal.
      Float = f; return *this;
228             }
```

Here is the call graph for this function:



#### 8.5.4.24 operator=() [6/6]

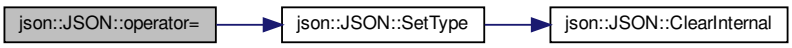
```
template<typename T >
enable_if<is_convertible<T,string>::value, JSON&>::type json::JSON::operator= (
            T s )  [inline]
```

Definition at line 231 of file json.hpp.

References Internal, SetType(), json::JSON::BackingData::String, and String.

```
231                                                                                 {
232                 SetType( Class::String ); *Internal.
      String = string( s ); return *this;
233             }
```

Here is the call graph for this function:




#### 8.5.4.25 operator[]() [1/2]

```
JSON& json::JSON::operator[] (
            const string & key )  [inline]
```

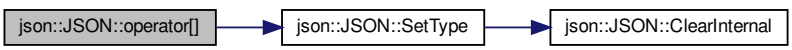
Definition at line 235 of file json.hpp.

References Internal, json::JSON::BackingData::Map, Object, and SetType().

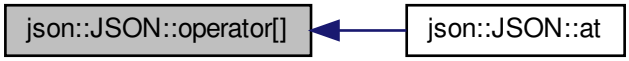
Referenced by at().

```
235                                                 {
236             SetType( Class::Object ); return Internal.
      Map->operator[]( key );
237         }
```

Here is the call graph for this function:



Here is the caller graph for this function:




#### 8.5.4.26 operator[]() [2/2]

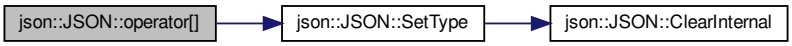
```
JSON& json::JSON::operator[] (
            unsigned index )  [inline]
```

Definition at line 239 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, and SetType().

```
239                                         {
240             SetType( Class::Array );
241             if( index >= Internal.List->size() ) Internal.
      List->resize( index + 1 );
242             return Internal.List->operator[]( index );
243         }
```

Here is the call graph for this function:




#### 8.5.4.27 SetType()

```
void json::JSON::SetType (
            Class type )  [inline], [private]
```

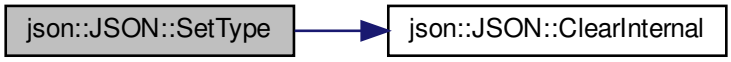
Definition at line 383 of file json.hpp.

References Array, json::JSON::BackingData::Bool, Boolean, ClearInternal(), json::JSON::BackingData::Float, Floating, json::JSON::BackingData::Int, Integral, Internal, json::JSON::BackingData::List, json::JSON::BackingData::↩
Map, Null, Object, json::JSON::BackingData::String, String, and Type.

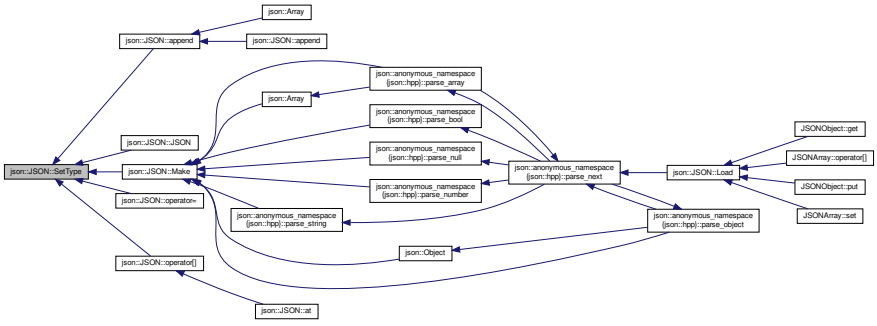
Referenced by append(), JSON(), Make(), operator=(), and operator[]().

```
383                                 {
384             if( type == Type )
385                 return;
386
387             ClearInternal();
388
389             switch( type ) {
390             case Class::Null:      Internal.Map    = nullptr;               break;
391             case Class::Object:    Internal.Map    = new map<string,JSON>(); break;
392             case Class::Array:     Internal.List   = new deque<JSON>();     break;
393             case Class::String:    Internal.String = new string();
    break;
394             case Class::Floating:  Internal.Float  = 0.0;
    break;
395             case Class::Integral:  Internal.Int    = 0;
    break;
396             case Class::Boolean:   Internal.Bool   = false;
    break;
397             }
398
399             Type = type;
400         }
```

Here is the call graph for this function:




Here is the caller graph for this function:




#### 8.5.4.28 size()

```
int json::JSON::size ( ) const  [inline]
```

Definition at line 274 of file json.hpp.

References Array, Internal, json::JSON::BackingData::List, json::JSON::BackingData::Map, Object, and Type.

```
274                         {
275             if( Type == Class::Object )
276                 return Internal.Map->size();
277             else if( Type == Class::Array )
278                 return Internal.List->size();
279             else
280                 return -1;
281         }
```

#### 8.5.4.29 ToBool() [1/2]

```
bool json::JSON::ToBool ( ) const  [inline]
```

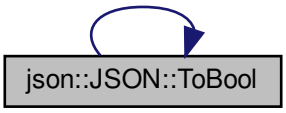
Definition at line 306 of file json.hpp.

References ToBool().

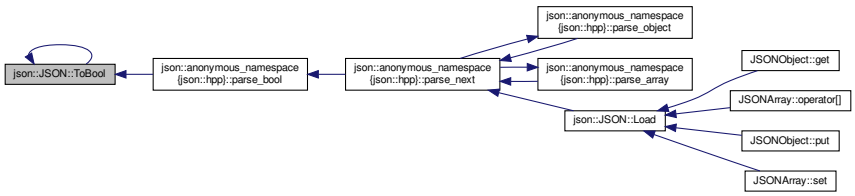
Referenced by json::anonymous_namespace{json.hpp}::parse_bool(), and ToBool().

```
306 { bool b; return ToBool( b ); }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 8.5.4.30 ToBool() [2/2]

```
bool json::JSON::ToBool (
            bool & ok ) const  [inline]
```

Definition at line 307 of file json.hpp.

References json::JSON::BackingData::Bool, Boolean, Internal, and Type.

```
307                                     {
308             ok = (Type == Class::Boolean);
309             return ok ? Internal.Bool : false;
310         }
```

#### 8.5.4.31 ToFloat() [1/2]

```
double json::JSON::ToFloat ( ) const  [inline]
```

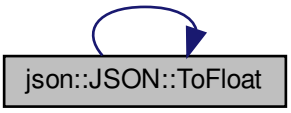
Definition at line 294 of file json.hpp.

References ToFloat().

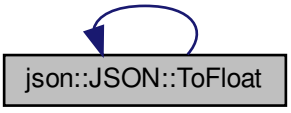
Referenced by ToFloat().

```
294 { bool b; return ToFloat( b ); }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 8.5.4.32 ToFloat() [2/2]

```
double json::JSON::ToFloat (
            bool & ok ) const  [inline]
```

Definition at line 295 of file json.hpp.

References json::JSON::BackingData::Float, Floating, Internal, and Type.

```
295                                     {
296             ok = (Type == Class::Floating);
297             return ok ? Internal.Float : 0.0;
298         }
```

#### 8.5.4.33 ToInt() [1/2]

```
long json::JSON::ToInt ( ) const  [inline]
```

Definition at line 300 of file json.hpp.

References ToInt().

Referenced by ToInt().

```
300 { bool b; return ToInt( b ); }
```

Here is the call graph for this function:


Here is the caller graph for this function:

#### 8.5.4.34 ToInt() [2/2]

```
long json::JSON::ToInt (
            bool & ok ) const  [inline]
```

Definition at line 301 of file json.hpp.

References json::JSON::BackingData::Int, Integral, Internal, and Type.

```
301                                     {
302             ok = (Type == Class::Integral);
303             return ok ? Internal.Int : 0;
304         }
```

#### 8.5.4.35 ToString() [1/2]

```
string json::JSON::ToString ( ) const  [inline]
```

Definition at line 288 of file json.hpp.

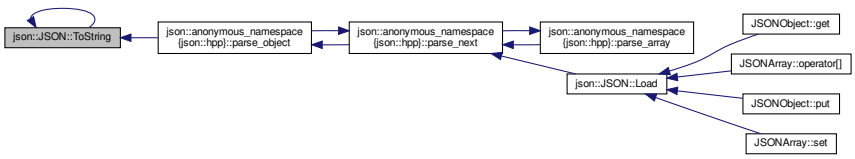
References ToString().

Referenced by json::anonymous_namespace{json.hpp}::parse_object(), and ToString().

```
288 { bool b; return std::move( ToString( b ) ); }
```

Here is the call graph for this function:


Here is the caller graph for this function:



#### 8.5.4.36 ToString() [2/2]

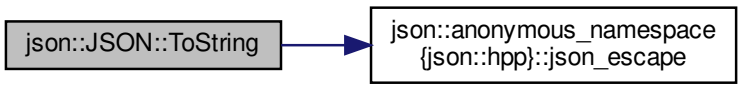
```
string json::JSON::ToString (
            bool & ok ) const  [inline]
```

Definition at line 289 of file json.hpp.

References Internal, json::anonymous_namespace{json.hpp}::json_escape(), json::JSON::BackingData::String, String, and Type.

```
289                                         {
290             ok = (Type == Class::String);
291             return ok ? std::move( json_escape( *Internal.
      String ) ): string("");
292         }
```

Here is the call graph for this function:




### 8.5.5 Friends And Related Function Documentation

#### 8.5.5.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const JSON & json )  [friend]
```

Definition at line 436 of file json.hpp.

```
436                                                                 {
437     os << json.dump();
438     return os;
439 }
```

### 8.5.6 Field Documentation

#### 8.5.6.1 Internal

```
union json::JSON::BackingData json::JSON::Internal  [private]
```

Referenced by append(), ArrayRange(), at(), ClearInternal(), dump(), hasKey(), JSON(), length(), ObjectRange(), operator=(), operator[ ](), SetType(), size(), ToBool(), ToFloat(), ToInt(), ToString(), and ~JSON().

#### 8.5.6.2 Type

```
Class json::JSON::Type = Class::Null  [private]
```

Definition at line 418 of file json.hpp.

Referenced by ArrayRange(), ClearInternal(), dump(), hasKey(), IsNull(), JSON(), JSONType(), length(), ObjectRange(), operator=(), SetType(), size(), ToBool(), ToFloat(), ToInt(), ToString(), and ~JSON().

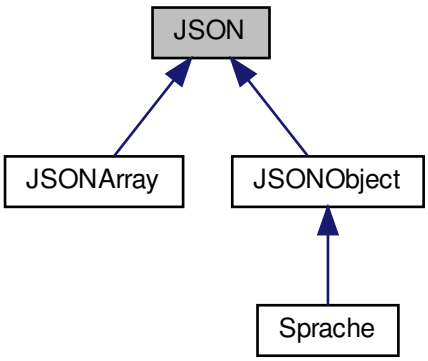
The documentation for this class was generated from the following file:

- lib/simplejson/json.hpp

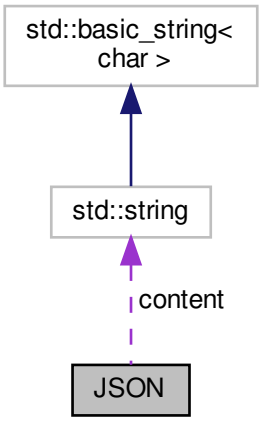
## 8.6 JSON Class Reference

```
#include <json.hpp>
```

Inheritance diagram for JSON:



Collaboration diagram for JSON:




### Public Member Functions

- JSON (std::string content)
- operator std::string ()

### Protected Attributes

- std::string content

### Friends

- std::ostream & operator<< (std::ostream &, JSON &)

### 8.6.1 Detailed Description

Klasse für den Arbeit mit JSON Dateien.

Definition at line 13 of file json.hpp.

### 8.6.2 Constructor & Destructor Documentation

#### 8.6.2.1 JSON()

```
JSON::JSON (
            std::string content )  [inline]
```

Konstruktor für die Klasse JSON. Ladet den `content` ein.

Definition at line 21 of file json.hpp.

```
21                                : content(content)
22         {}
```

### 8.6.3 Member Function Documentation

#### 8.6.3.1 operator std::string()

```
JSON::operator std::string ( )  [inline]
```

Gibt die internale Darstellung als eine String zurück..

Definition at line 26 of file json.hpp.

References content.

```
27         {return content;}
```

### 8.6.4 Friends And Related Function Documentation

#### 8.6.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            JSON & content )  [friend]
```

Operatorüberladung für den << operator, schreibt den Inhalt des JSON Klasses (also entweder eine JSONObject or eine JSONArray) zum ostream.

Definition at line 70 of file json.hpp.

```
71 {
72     os<<content.content;
73     return os;
74 }
```

### 8.6.5 Field Documentation

#### 8.6.5.1 content

```
std::string JSON::content  [protected]
```

Definition at line 15 of file json.hpp.

Referenced by JSONObject::get(), operator std::string(), operator<<(), JSONArray::operator[](), JSONObject::put(), and JSONArray::set().

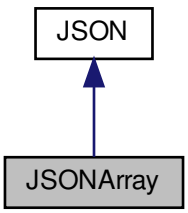
The documentation for this class was generated from the following file:

- src/json/json.hpp

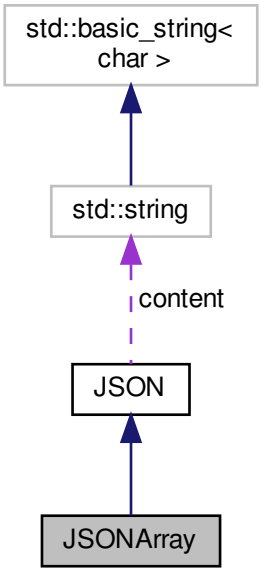
## 8.7 JSONArray Class Reference

```
#include <json.hpp>
```

Inheritance diagram for JSONArray:



Collaboration diagram for JSONArray:




### Public Member Functions

- JSONArray (std::string="[ ]")
- JSONObject operator[ ] (int n)
- template<typename T >
  void set (int, T)
- template<>
  void set (int n, JSON t)
- template<>
  void set (int key, JSONObject value)
- template<>
  void set (int key, JSONArray value)

### Additional Inherited Members

### 8.7.1 Detailed Description

Klasse für den Arbeit mit JSONArrays, ein Child von JSON

Definition at line 36 of file json.hpp.

### 8.7.2 Constructor & Destructor Documentation

#### 8.7.2.1 JSONArray()

```
JSONArray::JSONArray (
            std::string content = "[]" )  [inline]
```

Konstruktor für die Klasse JSONArray. Ladet den `content` ein.

Definition at line 152 of file json.hpp.

```
152                                             : JSON(content)
153 {}
```

### 8.7.3 Member Function Documentation

#### 8.7.3.1 operator[]()

```
JSONObject JSONArray::operator[] (
            int n )  [inline]
```

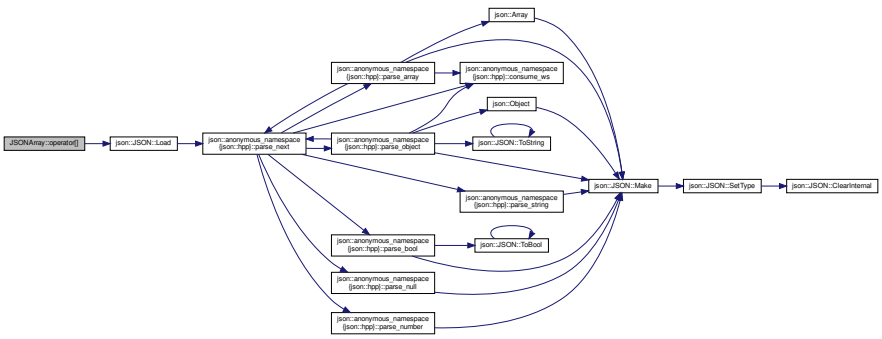
Gibt den Wert beim Platz n zurück.

Definition at line 158 of file json.hpp.

References JSON::content, and json::JSON::Load().

```
159 {
160     json::JSON obj = json::JSON::Load(content);
161     JSONObject json(obj[n].dump());
162     return json;
163 }
```

Here is the call graph for this function:



#### 8.7.3.2 set() [1/4]

```
template<typename T >
void JSONArray::set (
            int n,
            T t )  [inline]
```

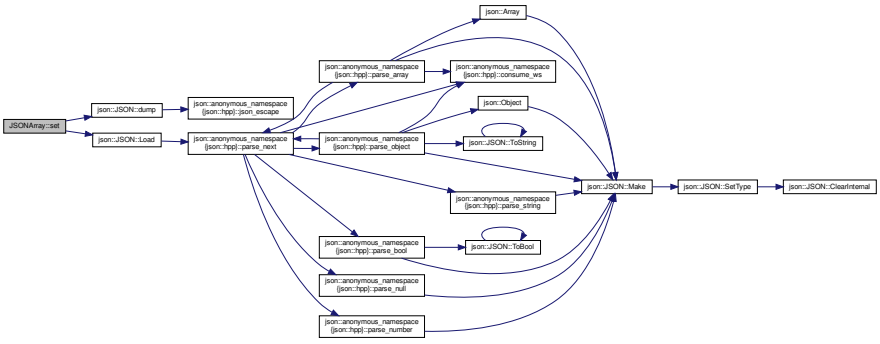
Einstellen den Wert beim Platz n. Der neue Wert wird t sein (Template).

Definition at line 168 of file json.hpp.

References JSON::content, json::JSON::dump(), and json::JSON::Load().

```
169 {
170         json::JSON obj = json::JSON::Load(content);
171         obj[n] = t;
172         content = obj.dump();
173 }
```

Here is the call graph for this function:




#### 8.7.3.3 set() [2/4]

```
template<>
void JSONArray::set (
            int n,
            JSON t )  [inline]
```

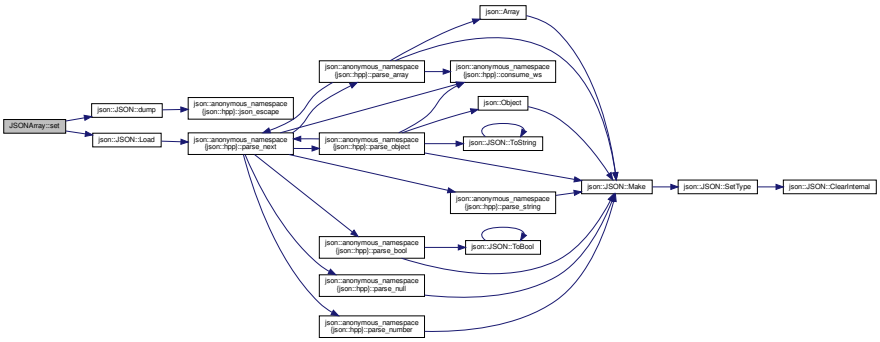
Einstellen den Wert beim Platz n. Der neue Wert wird t sein (JSON).

Definition at line 179 of file json.hpp.

References json::JSON::dump(), and json::JSON::Load().

```
180 {
181         json::JSON obj = json::JSON::Load(content);
182         json::JSON obj2 = json::JSON::Load((std::string)t);
183         obj[n] = obj2;
184         content = obj.dump();
185 }
```

Here is the call graph for this function:




#### 8.7.3.4 set() [3/4]

```
template<>
void JSONArray::set (
             int key,
             JSONObject value )  [inline]
```

Einstellen den Wert beim Platz n. Der neue Wert wird t sein (JSONObject).

Definition at line 191 of file json.hpp.

```
192 {
193     (*this).set(key, dynamic_cast<JSON&>(value));
194 }
```

#### 8.7.3.5 set() [4/4]

```
template<>
void JSONArray::set (
             int key,
             JSONArray value )  [inline]
```

Einstellen den Wert beim Platz n. Der neue Wert wird t sein (JSONArray).

Definition at line 199 of file json.hpp.

```
200 {
201     (*this).set(key, dynamic_cast<JSON&>(value));
202 }
```

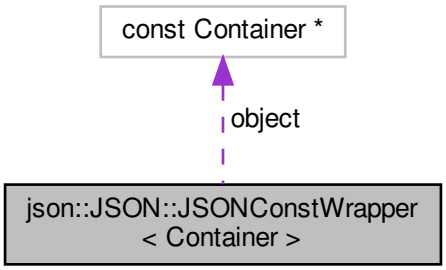
The documentation for this class was generated from the following file:

- src/json/json.hpp

## 8.8 json::JSON::JSONConstWrapper< Container > Class Template Reference

`#include <json.hpp>`

Collaboration diagram for json::JSON::JSONConstWrapper< Container >:




### Public Member Functions

- JSONConstWrapper (const Container *val)
- JSONConstWrapper (std::nullptr_t)
- Container::const_iterator begin () const
- Container::const_iterator end () const

### Private Attributes

- const Container * object

### 8.8.1 Detailed Description

**template<typename Container>**
**class json::JSON::JSONConstWrapper< Container >**

Definition at line 88 of file json.hpp.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 JSONConstWrapper() [1/2]

```
template<typename Container >
json::JSON::JSONConstWrapper< Container >::JSONConstWrapper (
            const Container * val )  [inline]
```

Definition at line 92 of file json.hpp.

```
92 : object( val ) {}
```

#### 8.8.2.2 JSONConstWrapper() [2/2]

```
template<typename Container >
json::JSON::JSONConstWrapper< Container >::JSONConstWrapper (
            std::nullptr_t  )  [inline]
```

Definition at line 93 of file json.hpp.

```
93 : object( nullptr ) {}
```

### 8.8.3 Member Function Documentation

#### 8.8.3.1 begin()

```
template<typename Container >
Container::const_iterator json::JSON::JSONConstWrapper< Container >::begin ( ) const  [inline]
```

Definition at line 95 of file json.hpp.

```
95 { return object ? object->begin() : typename Container::const_iterator(); }
```

#### 8.8.3.2 end()

```
template<typename Container >
Container::const_iterator json::JSON::JSONConstWrapper< Container >::end ( ) const  [inline]
```

Definition at line 96 of file json.hpp.

```
96 { return object ? object->end() : typename Container::const_iterator(); }
```

### 8.8.4 Field Documentation

#### 8.8.4.1 object

```
template<typename Container >
const Container* json::JSON::JSONConstWrapper< Container >::object  [private]
```

Definition at line 89 of file json.hpp.

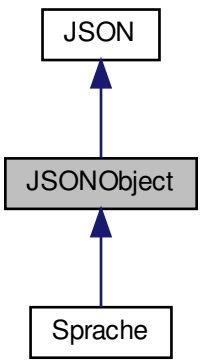
The documentation for this class was generated from the following file:

- lib/simplejson/json.hpp

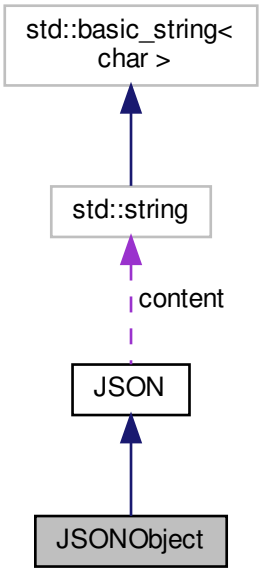
## 8.9 JSONObject Class Reference

```
#include <json.hpp>
```

Inheritance diagram for JSONObject:



Collaboration diagram for JSONObject:




## Public Member Functions

- JSONObject (std::string="{}")
- JSONObject get (std::string)
- template<typename T >
  void put (std::string, T)
- operator int ()
- JSONArray & toArr ()
- JSONObject operator[ ] (int)
- template<>
  void put (std::string key, JSON value)
- template<>
  void put (std::string key, JSONObject value)
- template<>
  void put (std::string key, JSONArray value)

## Additional Inherited Members

### 8.9.1 Detailed Description

Klasse für den Arbeit mit JSONObjects, ein Child von JSON

Definition at line 48 of file json.hpp.

### 8.9.2 Constructor & Destructor Documentation

#### 8.9.2.1 JSONObject()

```
JSONObject::JSONObject (
            std::string content = "{}" )  [inline]
```

Konstruktor des JSONObjectes, ladet den inhalt ein.

Definition at line 88 of file json.hpp.

```
88                                              : JSON(content)
89 {}
```

### 8.9.3 Member Function Documentation

#### 8.9.3.1 get()

```
JSONObject JSONObject::get (
            std::string key )  [inline]
```

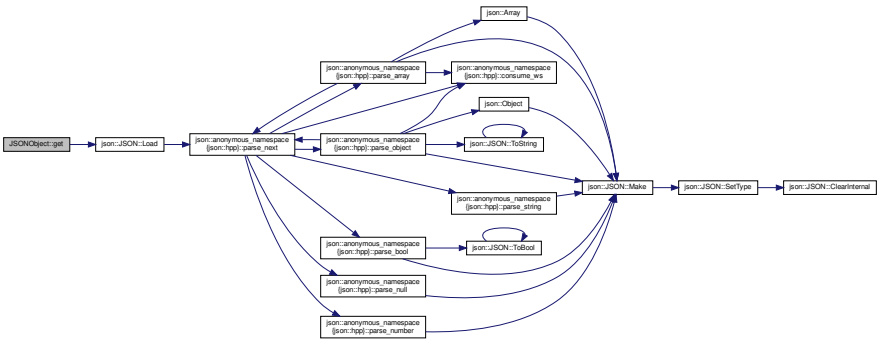
Gibt den Wert mit Schlüssel `key` zurück als eine neue JSONObject.

Definition at line 94 of file json.hpp.

References JSON::content, and json::JSON::Load().

```
95 {
96     json::JSON obj = json::JSON::Load(content);
97     JSONObject json(obj[key].dump());
98     return json;
99 }
```

Here is the call graph for this function:



#### 8.9.3.2 operator int()

```
JSONObject::operator int ( )  [inline]
```

Gibt den Wert des Objektes als ein Integer zurück.

Definition at line 82 of file json.hpp.

```
83 {return atoi(content.c_str());}
```

#### 8.9.3.3 operator[]()

```
JSONObject JSONObject::operator[] (
            int  )
```

#### 8.9.3.4 put() [1/4]

```
template<typename T >
void JSONObject::put (
            std::string key,
            T value )  [inline]
```

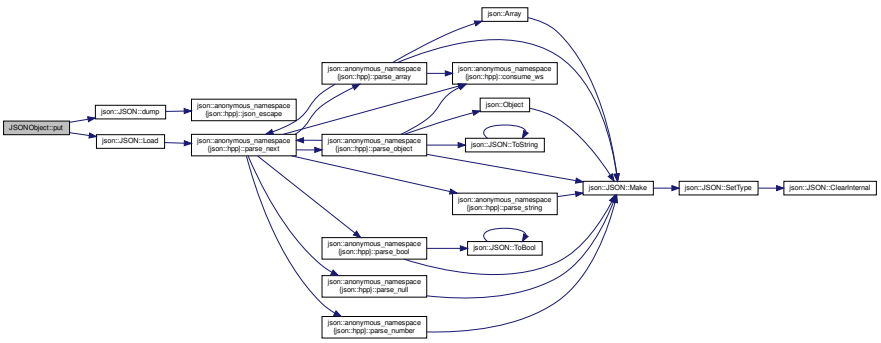
Ein neues Wert zum JSON Dateien einfügen mit Schlüssel `key` und Wert `value`. Diese Methode funktioniert für fast alle Werte (deswegen eine Template wird benutzt).

Definition at line 113 of file json.hpp.

References JSON::content, json::JSON::dump(), and json::JSON::Load().

```
114 {
115     json::JSON obj = json::JSON::Load(content);
116     obj[key] = value;
117     content = obj.dump();
118 }
```

Here is the call graph for this function:



#### 8.9.3.5 put() [2/4]

```
template<>
void JSONObject::put (
            std::string key,
            JSON value )  [inline]
```

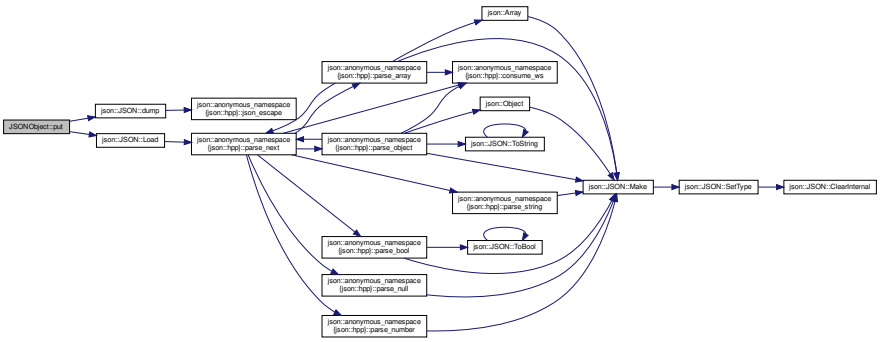
Ein neues Wert zum JSON Dateien einfügen mit Schlüssel `key` und Wert `value`. Diese Methode funktioniert nur für Werte mit JSON-Typ.

Definition at line 123 of file json.hpp.

References json::JSON::dump(), and json::JSON::Load().

```
124 {
125     json::JSON obj = json::JSON::Load(content);
126     json::JSON obj2 = json::JSON::Load((std::string)value);
127     obj[key] = obj2;
128     content = obj.dump();
129 }
```

Here is the call graph for this function:




#### 8.9.3.6 put() [3/4]

```
template<>
void JSONObject::put (
            std::string key,
            JSONObject value )  [inline]
```

Ein neues Wert zum JSON Dateien einfügen mit Schlüssel `key` und Wert `value`. Diese Methode funktioniert nur für JSONObjects.

Definition at line 135 of file json.hpp.

```
136 {
137     (*this).put(key, dynamic_cast<JSON&>(value));
138 }
```

#### 8.9.3.7 put() [4/4]

```
template<>
void JSONObject::put (
            std::string key,
            JSONArray value )  [inline]
```

Ein neues Wert zum JSON Dateien einfügen mit Schlüssel `key` und Wert `value`. Diese Methode funktioniert nur für JSONArrays.

Definition at line 143 of file json.hpp.

```
144 {
145     (*this).put(key, dynamic_cast<JSON&>(value));
146 }
```

#### 8.9.3.8 toArr()

```
JSONArray & JSONObject::toArr ( )  [inline]
```

Falls die JSONObject eine Array enthalten soll, muss man kastieren.

Definition at line 104 of file json.hpp.

```
105 {
106     return *static_cast<JSONArray*>(static_cast<JSON*>(this));
107 }
```

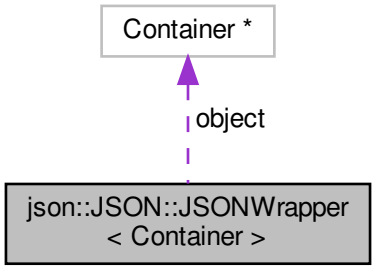
The documentation for this class was generated from the following file:

- src/json/json.hpp

## 8.10 json::JSON::JSONWrapper< Container > Class Template Reference

`#include <json.hpp>`

Collaboration diagram for json::JSON::JSONWrapper< Container >:



### Public Member Functions

- JSONWrapper (Container *val)
- JSONWrapper (std::nullptr_t)
- Container::iterator begin ()
- Container::iterator end ()
- Container::const_iterator begin () const
- Container::const_iterator end () const

### Private Attributes

- Container * object

## 8.10.1 Detailed Description

**template<typename Container>**
**class json::JSON::JSONWrapper< Container >**

Definition at line 74 of file json.hpp.

## 8.10.2 Constructor & Destructor Documentation

### 8.10.2.1 JSONWrapper() [1/2]

```
template<typename Container >
json::JSON::JSONWrapper< Container >::JSONWrapper (
            Container * val )  [inline]
```

Definition at line 78 of file json.hpp.

```
78 : object( val ) {}
```

### 8.10.2.2 JSONWrapper() [2/2]

```
template<typename Container >
json::JSON::JSONWrapper< Container >::JSONWrapper (
            std::nullptr_t  )  [inline]
```

Definition at line 79 of file json.hpp.

```
79 : object( nullptr ) {}
```

### 8.10.3 Member Function Documentation

#### 8.10.3.1 begin() [1/2]

```
template<typename Container >
Container::iterator json::JSON::JSONWrapper< Container >::begin ( )  [inline]
```

Definition at line 81 of file json.hpp.

```
81 { return object ? object->begin() : typename Container::iterator(); }
```

#### 8.10.3.2 begin() [2/2]

```
template<typename Container >
Container::const_iterator json::JSON::JSONWrapper< Container >::begin ( ) const  [inline]
```

Definition at line 83 of file json.hpp.

```
83 { return object ? object->begin() : typename Container::iterator(); }
```

#### 8.10.3.3 end() [1/2]

```
template<typename Container >
Container::iterator json::JSON::JSONWrapper< Container >::end ( )  [inline]
```

Definition at line 82 of file json.hpp.

```
82 { return object ? object->end() : typename Container::iterator(); }
```

#### 8.10.3.4 end() [2/2]

```
template<typename Container >
Container::const_iterator json::JSON::JSONWrapper< Container >::end ( ) const  [inline]
```

Definition at line 84 of file json.hpp.

```
84 { return object ? object->end() : typename Container::iterator(); }
```

### 8.10.4 Field Documentation

#### 8.10.4.1 object

```
template<typename Container >
Container* json::JSON::JSONWrapper< Container >::object  [private]
```

Definition at line 75 of file json.hpp.

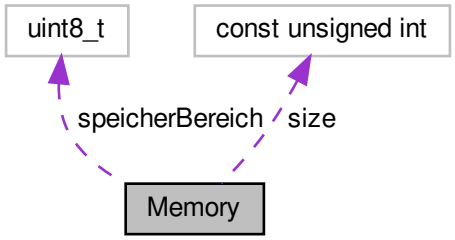
The documentation for this class was generated from the following file:

- lib/simplejson/json.hpp

## 8.11 Memory Class Reference

```
#include <memory.hpp>
```

Collaboration diagram for Memory:




### Public Member Functions

- const unsigned int getSize ()
- Memory (unsigned int)
- ~Memory ()
- Memory (const Memory &)
- bool write (unsigned int, uint8_t)
- uint8_t & read (unsigned int)
- bool shiftRight (unsigned int, uint8_t, uint8_t, uint8_t &, uint8_t)
- bool shiftLeft (unsigned int, uint8_t, uint8_t, uint8_t &, uint8_t)

### Private Attributes

- uint8_t * speicherBereich
- const unsigned int size

## 8.11.1 Detailed Description

Klasse für die Darstellung des Speicherplatzes.

Definition at line 11 of file memory.hpp.

## 8.11.2 Constructor & Destructor Documentation

### 8.11.2.1 Memory() [1/2]

```
Memory::Memory (
            unsigned int n )
```

Konstruktor des Speicherplatzes.

Definition at line 7 of file memory.cpp.

References speicherBereich.

```
7                               : size(n)
8 {
9     speicherBereich = new uint8_t[n];
10 }
```

### 8.11.2.2 ~Memory()

```
Memory::~Memory ( )
```

Destruktor des Speicherplatzes..

Definition at line 25 of file memory.cpp.

References speicherBereich.

```
26 {
27     delete [] speicherBereich;
28 }
```

#### 8.11.2.3 Memory() [2/2]

```
Memory::Memory (
            const Memory & obj )
```

Kopierkonstruktor des Speicherplatzes.

Definition at line 15 of file memory.cpp.

References size, and speicherBereich.

```
15                                : size(obj.size)
16 {
17     speicherBereich = new uint8_t[size];
18     memcpy(speicherBereich, obj.speicherBereich,
       size);
19 }
```

### 8.11.3 Member Function Documentation

#### 8.11.3.1 getSize()

```
const unsigned int Memory::getSize ( )
```

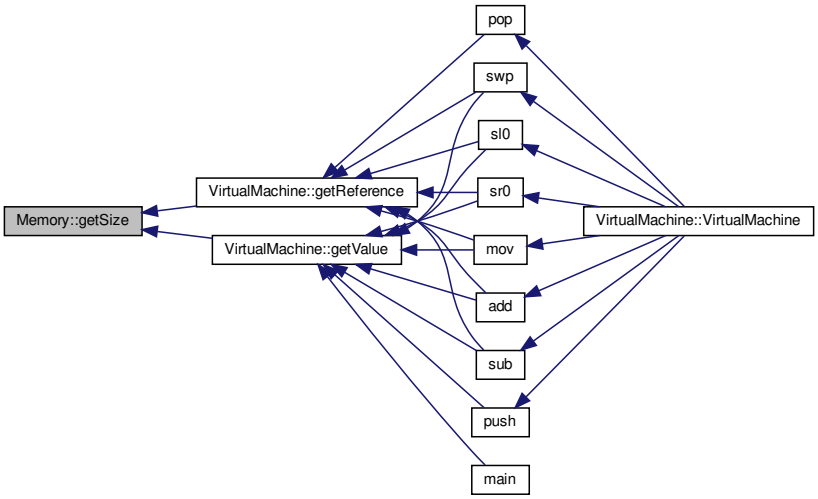
Definition at line 31 of file memory.cpp.

References size.

Referenced by VirtualMachine::getReference(), and VirtualMachine::getValue().

```
32 {
33     return size;
34 }
```

Here is the caller graph for this function:



#### 8.11.3.2 read()

```
uint8_t & Memory::read (
            unsigned int address )
```

Gibt den Wert beim `address` zurück.

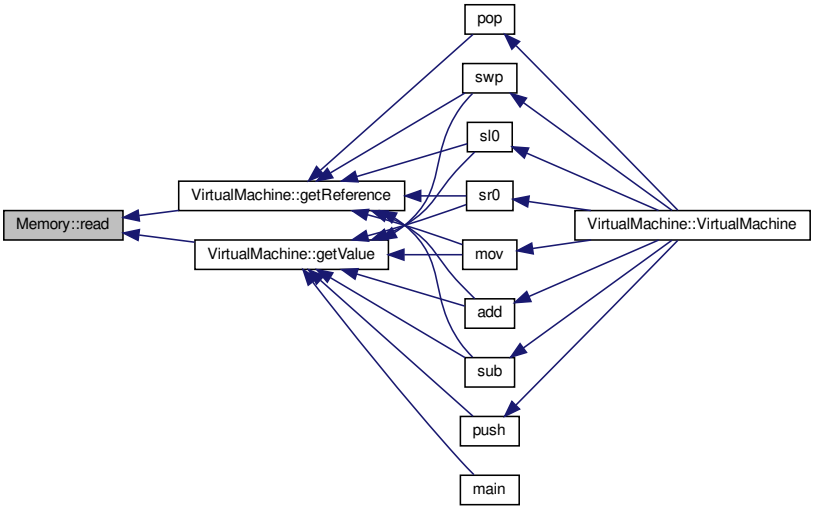
Definition at line 51 of file memory.cpp.

References speicherBereich.

Referenced by VirtualMachine::getReference(), and VirtualMachine::getValue().

```
52 {
53     return speicherBereich[address];
54 }
```

Here is the caller graph for this function:




#### 8.11.3.3 shiftLeft()

```
bool Memory::shiftLeft (
            unsigned int address,
            uint8_t data,
            uint8_t dataMask,
            uint8_t & flag,
            uint8_t mask )
```

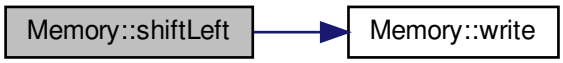
Verschiebt den Wert beim `address` nach links.

Definition at line 73 of file memory.cpp.

References size, speicherBereich, and write().

```
74 {
75     if(address >= size) return false;
76     uint8_t newdata = ((data & dataMask) && 1) | ((speicherBereich[address] & 0x7F) << 1);
77     flag &= ~mask;
78     flag |= ((speicherBereich[address] & 0x80)) ? mask : 0;
79     return this->write(address, newdata);
80
81
82 }
```

Here is the call graph for this function:




#### 8.11.3.4 shiftRight()

```
bool Memory::shiftRight (
              unsigned int address,
              uint8_t data,
              uint8_t dataMask,
              uint8_t & flag,
              uint8_t mask )
```

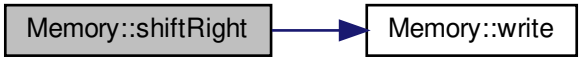
Verschiebt den Wert beim `address` nach rechts.

Definition at line 59 of file memory.cpp.

References size, speicherBereich, and write().

```
60 {
61     if(address >= size) return false;
62     uint8_t newdata = ((data & dataMask)&&1)<<7 | ((speicherBereich[address] & 0xFE) >> 1);
63     flag&= ~mask;
64     flag |= (speicherBereich[address] & 0x01) ? mask : 0;
65     return this->write(address, newdata);
66
67 }
```

Here is the call graph for this function:



#### 8.11.3.5 write()

```
bool Memory::write (
            unsigned int address,
            uint8_t data )
```

Schreibt data in dem Speicher beim `address` ein.

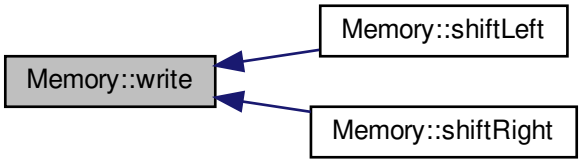
Definition at line 39 of file memory.cpp.

References size, and speicherBereich.

Referenced by shiftLeft(), and shiftRight().

```
40 {
41     if(address >= size) return false;
42     speicherBereich[address] = data;
43     return true;
44
45 }
```

Here is the caller graph for this function:




### 8.11.4 Field Documentation

#### 8.11.4.1 size

```
const unsigned int Memory::size  [private]
```

Definition at line 14 of file memory.hpp.

Referenced by getSize(), Memory(), shiftLeft(), shiftRight(), and write().

#### 8.11.4.2 speicherBereich

```
uint8_t* Memory::speicherBereich  [private]
```

Definition at line 13 of file memory.hpp.

Referenced by Memory(), read(), shiftLeft(), shiftRight(), write(), and ~Memory().

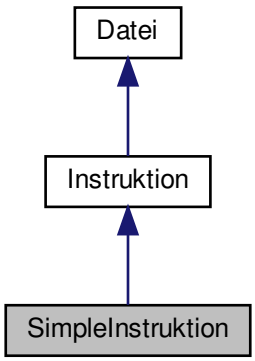
The documentation for this class was generated from the following files:

- src/memory/memory.hpp
- src/memory/memory.cpp

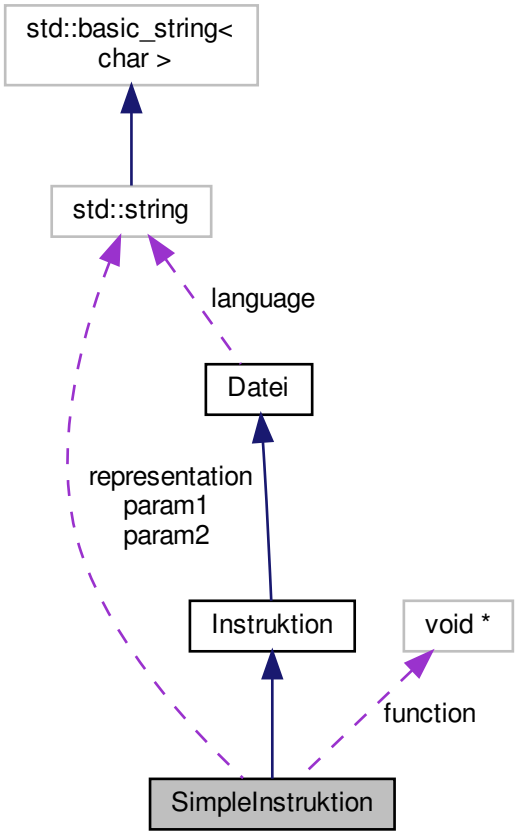
## 8.12 SimpleInstruktion Class Reference

```
#include <instruction.hpp>
```

Inheritance diagram for SimpleInstruktion:



Collaboration diagram for SimpleInstruktion:




### Public Member Functions

- SimpleInstruktion (std::string, std::string, void ∗, std::string, std::string)
- std::string print ()
- void run (VirtualMachine &)

### Private Attributes

- std::string representation
- void ∗ function
- std::string param1
- std::string param2

### Additional Inherited Members

#### 8.12.1 Detailed Description

Klasse für eine einzige Instruktion.

Definition at line 25 of file instruction.hpp.

### 8.12.2 Constructor & Destructor Documentation

#### 8.12.2.1 SimpleInstruktion()

```
SimpleInstruktion::SimpleInstruktion (
              std::string lang,
              std::string repr,
              void * ptr,
              std::string p1,
              std::string p2 )
```

Konstruktor der SimpleInstruktion Klasse und initialisiert die Membervariablen Sprache, representation, function und die 2 Parametern.

Definition at line 8 of file instruction.cpp.

```
8                                                                                                    :
       Instruktion(lang), representation(repr), function(ptr),
      param1(p1), param2(p2)
9 {
10     this->function = ptr;
11 }
```

### 8.12.3 Member Function Documentation

#### 8.12.3.1 print()

```
std::string SimpleInstruktion::print ( )  [virtual]
```

Gibt eine String-Representation zurück.

Implements Instruktion.

Definition at line 16 of file instruction.cpp.

References representation.

```
17 {
18     return representation;
19 }
```

#### 8.12.3.2 run()

```
void SimpleInstruktion::run (
            VirtualMachine & vm )  [virtual]
```

ausführt den Inhaltinstruktion durch eine Funktionenpointer zum entsprechenden Funktion.

Implements Instruktion.

Definition at line 25 of file instruction.cpp.

References param1, and param2.

```
26 {
27     void (*ptr)(VirtualMachine&, std::string, std::string) = (void (*)(
      VirtualMachine&, std::string, std::string))function;
28     (*ptr)(vm, param1, param2);
29 }
```

### 8.12.4 Field Documentation

#### 8.12.4.1 function

```
void* SimpleInstruktion::function  [private]
```

Definition at line 29 of file instruction.hpp.

#### 8.12.4.2 param1

```
std::string SimpleInstruktion::param1  [private]
```

Definition at line 30 of file instruction.hpp.

Referenced by run().

#### 8.12.4.3 param2

```
std::string SimpleInstruktion::param2  [private]
```

Definition at line 31 of file instruction.hpp.

Referenced by run().

#### 8.12.4.4 representation

```
std::string SimpleInstruktion::representation  [private]
```

Definition at line 28 of file instruction.hpp.

Referenced by print().

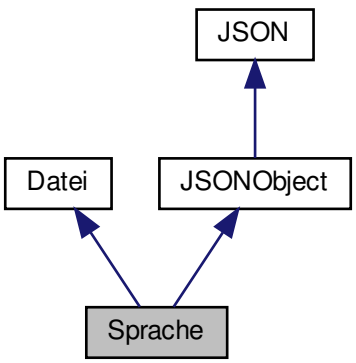
The documentation for this class was generated from the following files:

- src/datei/instruction/instruction.hpp
- src/datei/instruction/instruction.cpp

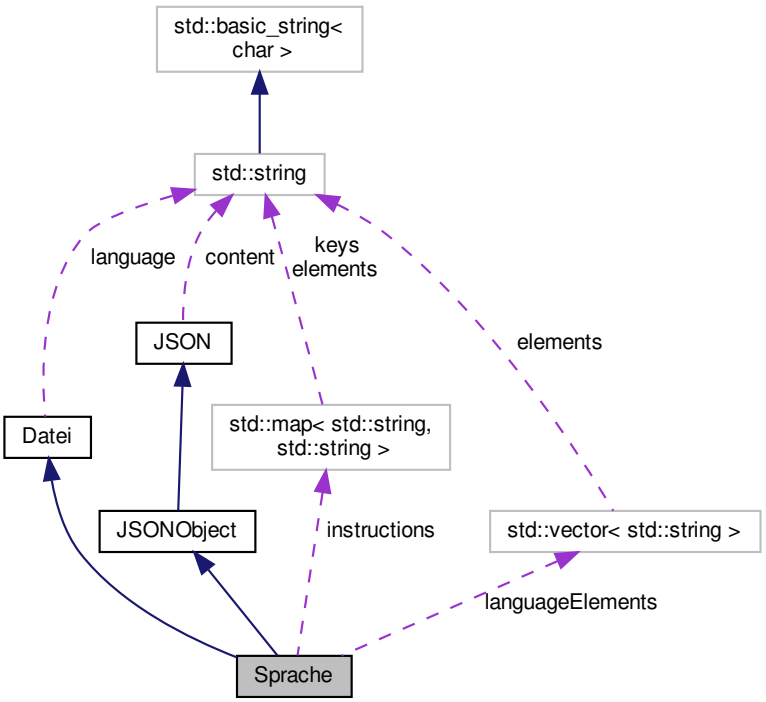
## 8.13 Sprache Class Reference

```
#include <sprache.hpp>
```

Inheritance diagram for Sprache:



Collaboration diagram for Sprache:




## Public Member Functions

- Sprache (std::string, std::string)
- std::string print ()

## Data Fields

- std::map< std::string, std::string > instructions

## Static Public Attributes

- static std::vector< std::string > languageElements

## Additional Inherited Members

### 8.13.1 Detailed Description

Child von 2 Basisklassen. Enthielt die Definitionen des Befehle.

Definition at line 14 of file sprache.hpp.

### 8.13.2 Constructor & Destructor Documentation

#### 8.13.2.1 Sprache()

```
Sprache::Sprache (
            std::string data,
            std::string lang )
```

Konstruktor für Klasse Sprache. Ordnet die Befehle zu ihnen Bedeutungen.

Definition at line 19 of file sprache.cpp.

```
19                                          : Datei(lang), JSONObject(data),
      instructions({
20     {this->get("Move B to A"), "Move B to A"},
21     {this->get("Add B to A"), "Add B to A"},
22     {this->get("Substract B from A"), "Substract B from A"},
23     {this->get("Swap the upper and lower 4 bits"), "Swap the upper and lower 4 bits"},
24     {this->get("Shift left, insert 0"), "Shift left, insert 0"},
25     {this->get("Shift right, insert 0"), "Shift right, insert 0"},
26     {this->get("Jump to subroutine"), "Jump to subroutine"},
27     {this->get("Push value to stack"), "Push value to stack"},
28     {this->get("Pop value from stack"), "Pop value from stack"},
29     })
30 {
31 }
```

### 8.13.3 Member Function Documentation

#### 8.13.3.1 print()

```
std::string Sprache::print ( )  [virtual]
```

Implements Datei.

Definition at line 34 of file sprache.cpp.

```
35 {
36     return (std::string)*this;
37 }
```

### 8.13.4 Field Documentation

#### 8.13.4.1 instructions

```
std::map<std::string, std::string> Sprache::instructions
```

Definition at line 17 of file sprache.hpp.

Referenced by VirtualMachine::getPtr(), and VirtualMachine::runInstruction().

#### 8.13.4.2 languageElements

```
std::vector< std::string > Sprache::languageElements  [static]
```

**Initial value:**

```
= {
    "Move B to A",
    "Add B to A",
    "Substract B from A",
    "Swap the upper and lower 4 bits",
    "Shift left, insert 0",
    "Shift right, insert 0",
    "Jump to subroutine",
    "Push value to stack",
    "Pop value from stack"
}
```

Definition at line 18 of file sprache.hpp.

Referenced by VirtualMachine::VirtualMachine().

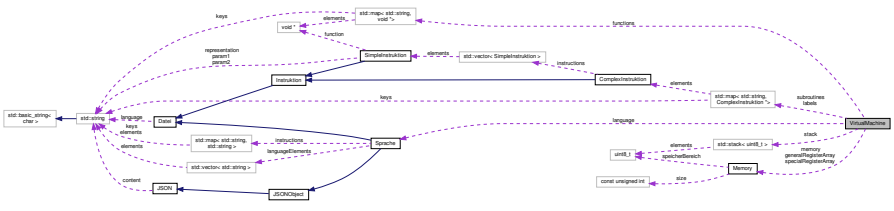
The documentation for this class was generated from the following files:

- src/datei/sprache/sprache.hpp
- src/datei/sprache/sprache.cpp

## 8.14 VirtualMachine Class Reference

```
#include <virtualmachine.hpp>
```

Collaboration diagram for VirtualMachine:



### Public Member Functions

- VirtualMachine (Sprache, unsigned int=1024, unsigned int=16)
- bool runInstruction (std::string)
- void reRunAll ()
- bool addSubroutine (std::string)
- uint8_t & getReference (std::string)
- uint8_t getValue (std::string)
- void runSubroutine (std::string)
- void pushValue (uint8_t)
- uint8_t popValue ()

### Private Member Functions

- void * getPtr (std::string)

### Private Attributes

- std::stack< uint8_t > stack
- Memory memory
- Memory generalRegisterArray
- Memory specialRegisterArray
- std::map< std::string, ComplexInstruktion * > labels
- std::map< std::string, ComplexInstruktion * > subroutines
- Sprache language
- std::map< std::string, void * > functions

#### 8.14.1 Detailed Description

Klasse für den virtuellen Gerät, wodurch die Instruktionen ausgeführt werden können.

Definition at line 18 of file virtualmachine.hpp.

#### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 VirtualMachine()

```
VirtualMachine::VirtualMachine (
            Sprache sprache,
            unsigned int memory = 1024,
            unsigned int general = 16 )
```

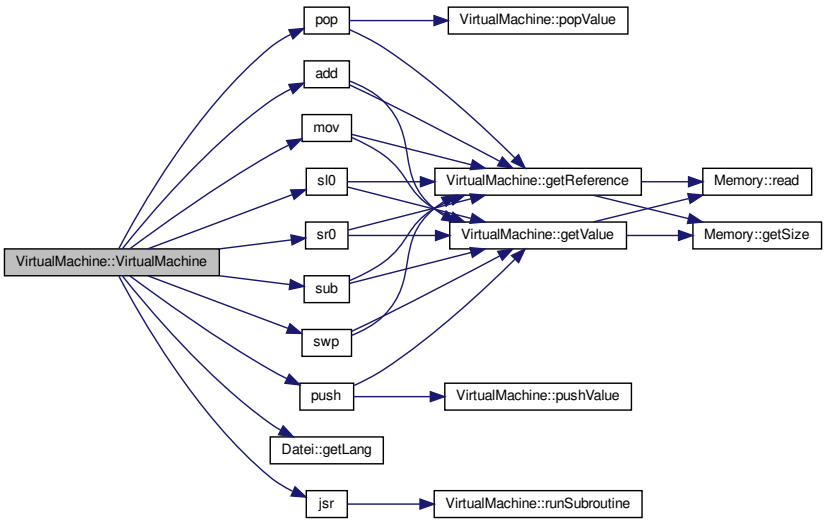
Konstruktor der Klasse VirtualMachine..

Definition at line 170 of file virtualmachine.cpp.

References add(), functions, Datei::getLang(), jsr(), language, Sprache::languageElements, mov(), pop(), push(), sl0(), sr0(), sub(), subroutines, and swp().

```
170                                                                                 :
      memory(memory), generalRegisterArray(general),
      specialRegisterArray(2), language(sprache)
171 {
172     std::vector<std::string>::iterator i = sprache.languageElements.begin();
173     functions.insert(std::make_pair( *(i++), (void*)mov));
174     functions.insert(std::make_pair( *(i++), (void*)add));
175     functions.insert(std::make_pair( *(i++), (void*)sub));
176     functions.insert(std::make_pair( *(i++), (void*)swp));
177     functions.insert(std::make_pair( *(i++), (void*)sl0));
178     functions.insert(std::make_pair( *(i++), (void*)sr0));
179     functions.insert(std::make_pair( *(i++), (void*)jsr));
180     functions.insert(std::make_pair( *(i++), (void*)push));
181     functions.insert(std::make_pair( *(i++), (void*)pop));
182
183     subroutines.insert(std::make_pair("_start",(new
      ComplexInstruktion(language.getLang()))));
184 }
```

Here is the call graph for this function:




### 8.14.3 Member Function Documentation

#### 8.14.3.1 addSubroutine()

```
bool VirtualMachine::addSubroutine (
            std::string s )
```

Addieren eine Subroutine

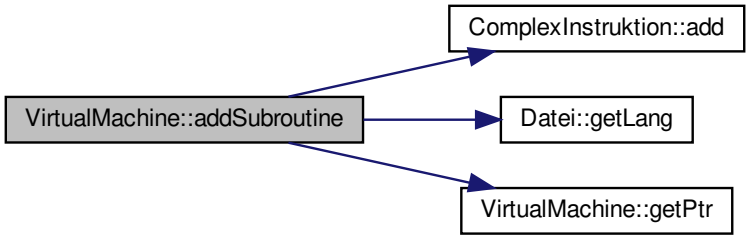
Definition at line 262 of file virtualmachine.cpp.

References ComplexInstruktion::add(), Datei::getLang(), getPtr(), language, and subroutines.

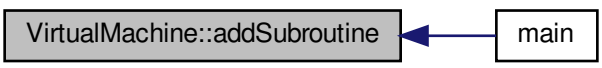
Referenced by main().

```
263 {
264     std::istringstream iss(s);
265     std::string buff;
266     ComplexInstruktion *ci = new ComplexInstruktion(
    language.getLang());
267     std::getline(iss, buff);
268     subroutines.insert(std::make_pair(buff, ci));
269     while (std::getline(iss, buff)) {
270         std::istringstream is(buff);
271         std::string instruction, param1, param2;
272         is>>instruction;
273         is>>param1;
274         is>>param2;
275         SimpleInstruktion *sNew = new SimpleInstruktion(
    language.getLang(), buff, getPtr(instruction), param1, param2);
276         ci->add(*sNew);
277     }
278 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 8.14.3.2 getPtr()

```
void * VirtualMachine::getPtr (
            std::string s )  [private]
```

Gibt den Funktionenpointer zum Befehl zurück..

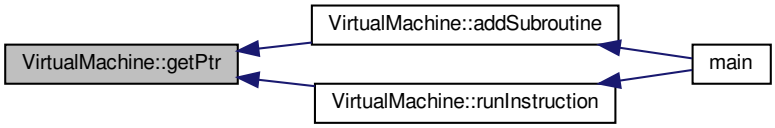
Definition at line 190 of file virtualmachine.cpp.

References functions, Sprache::instructions, and language.

Referenced by addSubroutine(), and runInstruction().

```
191 {
192     std::string str;
193     for(auto pair : language.instructions)
194     {
195         if(pair.first == s)
196         {
197             str = pair.second;
198             break;
199         }
200     }
201     for(auto pair : functions)
202     {
203         if(pair.first == str)
204         {
205             return (pair.second);
206         }
207     }
208 }
```

Here is the caller graph for this function:




#### 8.14.3.3 getReference()

```
uint8_t & VirtualMachine::getReference (
            std::string s )
```

Gibt den Referenz des Wertes zurück (also es geändert werden kann).

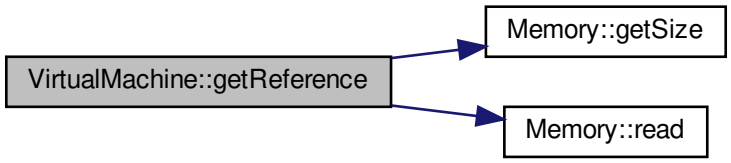
Definition at line 108 of file virtualmachine.cpp.

References generalRegisterArray, Memory::getSize(), memory, and Memory::read().

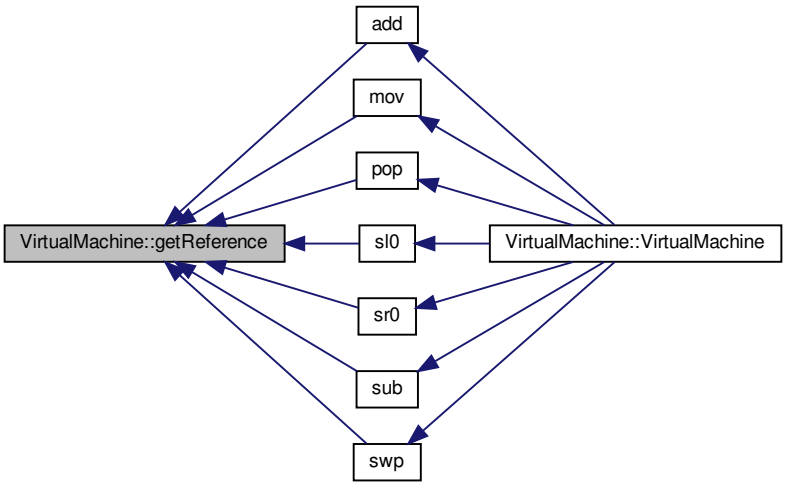
Referenced by add(), mov(), pop(), sl0(), sr0(), sub(), and swp().

```
109 {
110     unsigned int reg;
111     std::stringstream ss;
112     switch(s[0])
113     {
114         case '(':
115             ss << s.substr(2, s.size()-1);
116             ss >> reg;
117             if(generalRegisterArray.read(reg) > (this->
    memory).getSize()) throw ("Memory is out of bounds. Please cross-check syntax!");
118             return memory.read(generalRegisterArray.
    read(reg));
119             break;
120         case 'r':
121             ss << s.substr(1, s.size());
122             ss >> reg;
123             if(reg > generalRegisterArray.getSize()) throw ("Register does not
     exist. Please cross-check syntax!");
124             return generalRegisterArray.read(reg);
125             break;
126         default:
127             throw ("Please cross-check syntax!");
128             break;
129     }
130 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 8.14.3.4 getValue()

```
uint8_t VirtualMachine::getValue (
            std::string s )
```

Gibt den Wert zurück (nicht veränderbar).

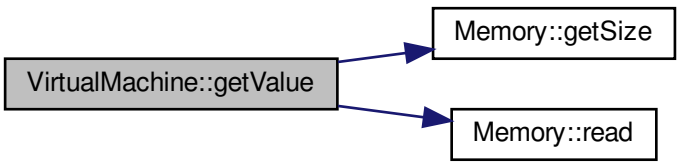
Definition at line 136 of file virtualmachine.cpp.

References generalRegisterArray, Memory::getSize(), memory, and Memory::read().

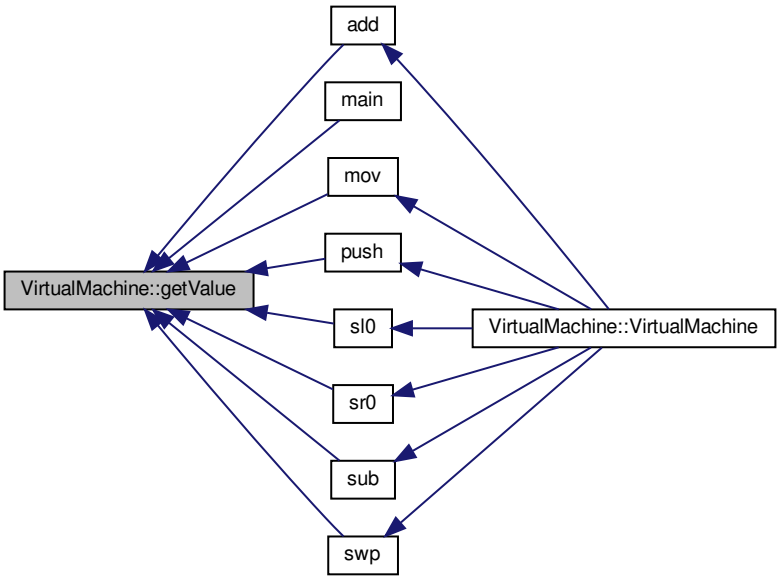
Referenced by add(), main(), mov(), push(), sl0(), sr0(), sub(), and swp().

```
137 {
138     unsigned int reg;
139     unsigned int ret;
140     std::stringstream ss;
141     std::istringstream iss(s);
142     switch(s[0])
143     {
144         case '(':
145             ss << s.substr(2, s.size()-1);
146             ss >> reg;
147             if(generalRegisterArray.read(reg) > (this->
    memory).getSize()) throw ("Memory is out of bounds. Please cross-check syntax!");
148             return memory.read(generalRegisterArray.
    read(reg));
149             break;
150         case 'r':
151             ss << s.substr(1, s.size());
152             ss >> reg;
153             if(reg > generalRegisterArray.getSize()) throw ("Register does not
     exist. Please cross-check syntax!");
154             return generalRegisterArray.read(reg);
155             break;
156         case '0':
157             iss >> std::hex >> ret;
158             return ret;
159         default:
160             throw ("Please cross-check syntax!");
161             break;
162     }
163
164 }
```

Here is the call graph for this function:



Here is the caller graph for this function:




#### 8.14.3.5 popValue()

```
uint8_t VirtualMachine::popValue ( )
```

Stackmanipulation.

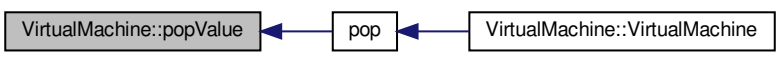
Definition at line 81 of file virtualmachine.cpp.

References stack.

Referenced by pop().

```
82 {
83     uint8_t t = stack.top();
84     stack.pop();
85     return t;
86 }
```

Here is the caller graph for this function:



#### 8.14.3.6 pushValue()

```
void VirtualMachine::pushValue (
            uint8_t t )
```

Stackmanipulation

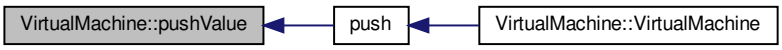
Definition at line 73 of file virtualmachine.cpp.

References stack.

Referenced by push().

```
74 {
75     stack.push(t);
76 }
```

Here is the caller graph for this function:




#### 8.14.3.7 reRunAll()

```
void VirtualMachine::reRunAll ( )
```

Ausführen jede Instruktion nocheinmal.

Definition at line 246 of file virtualmachine.cpp.

References subroutines.

Referenced by main().

```
247 {
248     for(auto pair : subroutines)
249         {
250             if(pair.first == "_start")
251             {
252 
253                 pair.second->run((*this));
254             }
255         }
256 }
```

Here is the caller graph for this function:

#### 8.14.3.8 runInstruction()

```
bool VirtualMachine::runInstruction (
            std::string r )
```

Parsen eine Instruktion aus einem String, zum VM history einfügen und danach laufen lassen

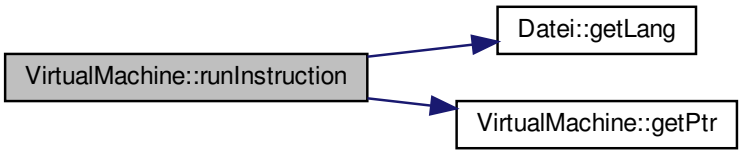
Definition at line 215 of file virtualmachine.cpp.

References Datei::getLang(), getPtr(), Sprache::instructions, language, and subroutines.

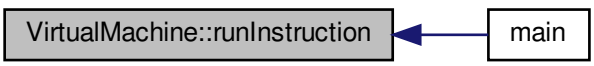
Referenced by main().

```
216 {
217     std::istringstream is(r);
218     std::string instruction, param1, param2;
219     is>>instruction;
220     bool is_ok = false;
221     for(auto i : language.instructions)
222     {
223         if(i.first == instruction) is_ok = true;
224     }
225     if(!is_ok) throw ("Please cross-check spelling of the mnemonik!");
226     is>>param1;
227     is>>param2;
228     SimpleInstruktion *sNew = new SimpleInstruktion(
      language.getLang(), r, getPtr(instruction), param1, param2);
229     for(std::pair<std::string, ComplexInstruktion*> pair : subroutines)
230     {
231         if(pair.first == "_start")
232         {
233
234             pair.second->add(*sNew);
235             break;
236         }
237     }
238     sNew -> run(*this);
239     return true;
240 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 8.14.3.9 runSubroutine()

```
void VirtualMachine::runSubroutine (
            std::string s )
```

Ausführt eine andere subroutine.

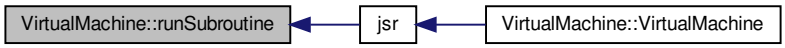
Definition at line 92 of file virtualmachine.cpp.

References subroutines.

Referenced by jsr().

```
93 {
94     for(auto pair : subroutines)
95     {
96         if(pair.first == s)
97         {
98
99             pair.second->run((*this));
100         }
101     }
102 }
```

Here is the caller graph for this function:




### 8.14.4 Field Documentation

#### 8.14.4.1 functions

```
std::map<std::string, void*> VirtualMachine::functions  [private]
```

Definition at line 28 of file virtualmachine.hpp.

Referenced by getPtr(), and VirtualMachine().

#### 8.14.4.2 generalRegisterArray

```
Memory VirtualMachine::generalRegisterArray  [private]
```

Definition at line 23 of file virtualmachine.hpp.

Referenced by getReference(), and getValue().

#### 8.14.4.3 labels

```
std::map<std::string, ComplexInstruktion*> VirtualMachine::labels  [private]
```

Definition at line 25 of file virtualmachine.hpp.

#### 8.14.4.4 language

```
Sprache VirtualMachine::language  [private]
```

Definition at line 27 of file virtualmachine.hpp.

Referenced by addSubroutine(), getPtr(), runInstruction(), and VirtualMachine().

#### 8.14.4.5 memory

```
Memory VirtualMachine::memory  [private]
```

Definition at line 22 of file virtualmachine.hpp.

Referenced by getReference(), and getValue().

#### 8.14.4.6 specialRegisterArray

```
Memory VirtualMachine::specialRegisterArray  [private]
```

Definition at line 24 of file virtualmachine.hpp.

#### 8.14.4.7 stack

```
std::stack<uint8_t> VirtualMachine::stack  [private]
```

Definition at line 21 of file virtualmachine.hpp.

Referenced by popValue(), and pushValue().

#### 8.14.4.8 subroutines

```
std::map<std::string, ComplexInstruktion*> VirtualMachine::subroutines  [private]
```

Definition at line 26 of file virtualmachine.hpp.

Referenced by addSubroutine(), reRunAll(), runInstruction(), runSubroutine(), and VirtualMachine().

The documentation for this class was generated from the following files:

- src/virtualmachine/virtualmachine.hpp
- src/virtualmachine/virtualmachine.cpp

# Chapter 9

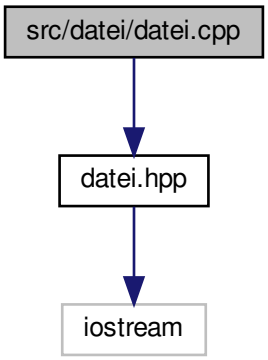
# File Documentation

## 9.1 docs/assets/datei.cpp File Reference

## 9.2 src/datei/datei.cpp File Reference

```
#include "datei.hpp"
```

Include dependency graph for datei.cpp:




### Functions

- std::ostream & operator<< (std::ostream &os, Datei &d)

### 9.2.1 Function Documentation

#### 9.2.1.1 operator<<()

```
std::ostream& operator<< (
             std::ostream & os,
             Datei & d )
```

Auschreiben.

Definition at line 13 of file datei.cpp.

References Datei::print().

```
14 {
15     os<<d.print();
16     return os;
17 }
```

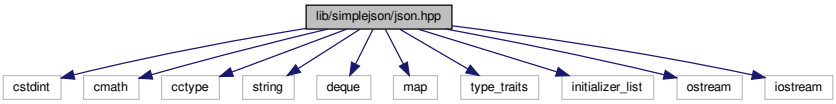
Here is the call graph for this function:


## 9.3 docs/spezifikation.md File Reference

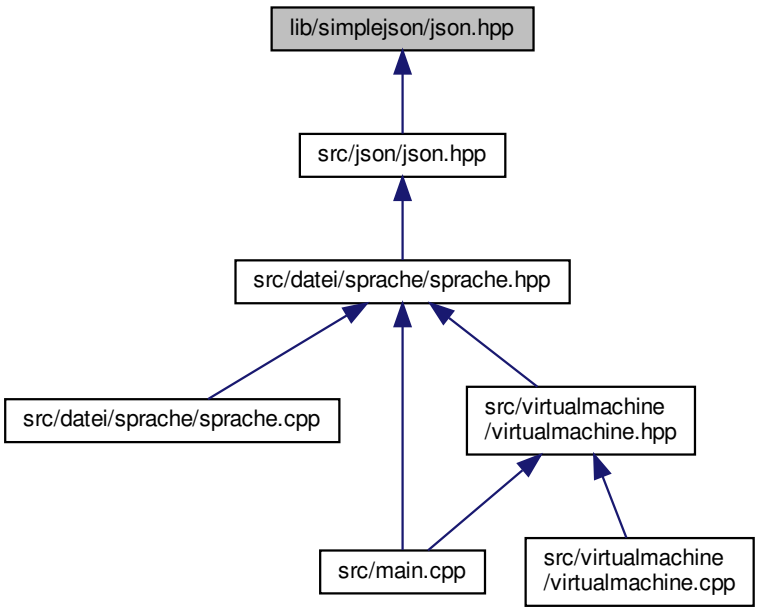
## 9.4 lib/simplejson/json.hpp File Reference

```
#include <cstdint>
#include <cmath>
#include <cctype>
#include <string>
#include <deque>
#include <map>
#include <type_traits>
#include <initializer_list>
#include <ostream>
#include <iostream>
```

Include dependency graph for json.hpp:



This graph shows which files directly or indirectly include this file:




## Data Structures

- class json::JSON
- union json::JSON::BackingData
- class json::JSON::JSONWrapper< Container >
- class json::JSON::JSONConstWrapper< Container >

## Namespaces

- json
- json::anonymous_namespace{json.hpp}

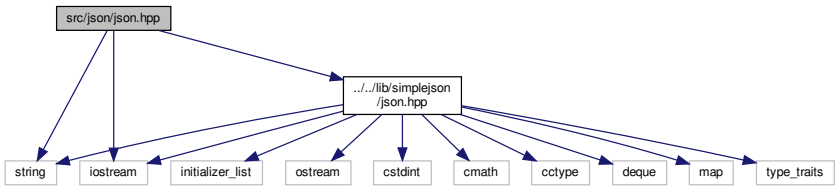
## Functions

- string json::anonymous_namespace{json.hpp}::json_escape (const string &str)
- JSON json::Array ()
- template<typename... T>
  JSON json::Array (T... args)
- JSON json::Object ()
- std::ostream & json::operator<< (std::ostream &os, const JSON &json)
- JSON json::anonymous_namespace{json.hpp}::parse_next (const string &, size_t &)
- void json::anonymous_namespace{json.hpp}::consume_ws (const string &str, size_t &offset)
- JSON json::anonymous_namespace{json.hpp}::parse_object (const string &str, size_t &offset)
- JSON json::anonymous_namespace{json.hpp}::parse_array (const string &str, size_t &offset)
- JSON json::anonymous_namespace{json.hpp}::parse_string (const string &str, size_t &offset)
- JSON json::anonymous_namespace{json.hpp}::parse_number (const string &str, size_t &offset)
- JSON json::anonymous_namespace{json.hpp}::parse_bool (const string &str, size_t &offset)
- JSON json::anonymous_namespace{json.hpp}::parse_null (const string &str, size_t &offset)

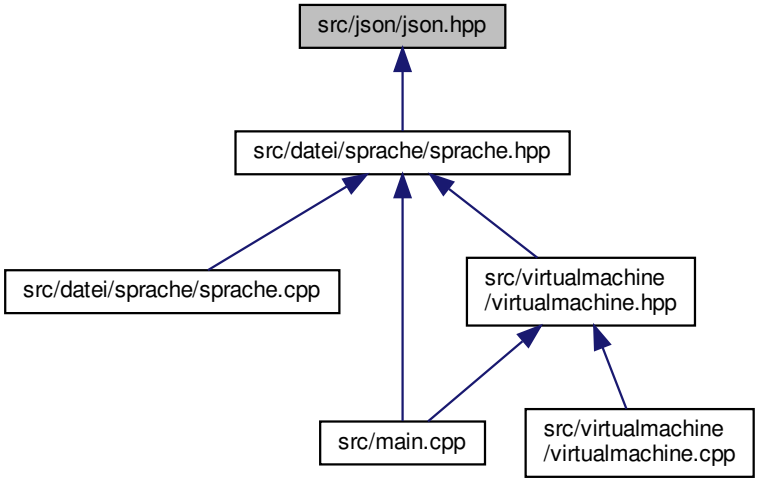
## 9.5 src/json/json.hpp File Reference

```
#include <string>
#include <iostream>
#include "../../lib/simplejson/json.hpp"
```

Include dependency graph for json.hpp:




This graph shows which files directly or indirectly include this file:




### Data Structures

- class JSON
- class JSONArray
- class JSONObject

### Functions

- std::ostream & operator<< (std::ostream &os, JSON &content)

### 9.5.1 Function Documentation

#### 9.5.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            JSON & content )  [inline]
```

Operatorüberladung für den << operator, schreibt den Inhalt des JSON Klasses (also entweder eine JSONObject or eine JSONArray) zum ostream.

Definition at line 70 of file json.hpp.

References JSON::content.

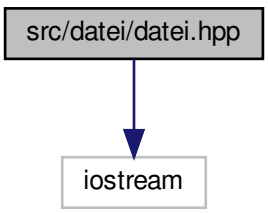
```
71 {
72     os<<content.content;
73     return os;
74 }
```

## 9.6 README.md File Reference

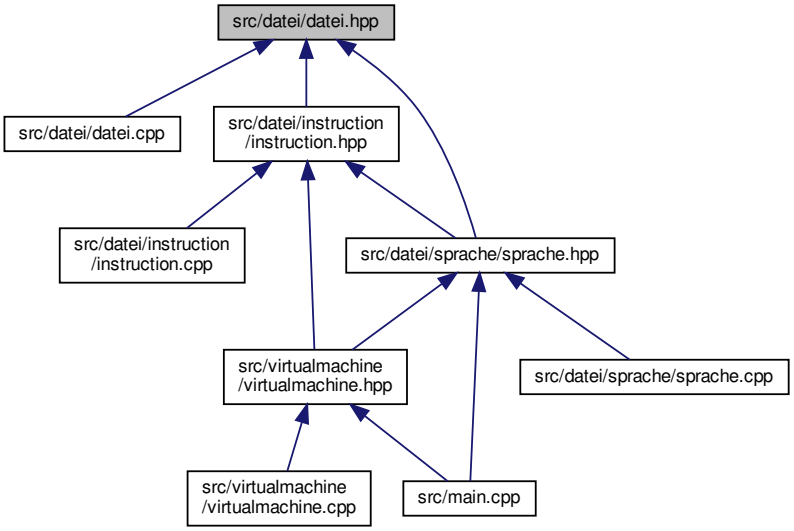
## 9.7 src/datei/datei.hpp File Reference

```
#include <iostream>
```

Include dependency graph for datei.hpp:



This graph shows which files directly or indirectly include this file:



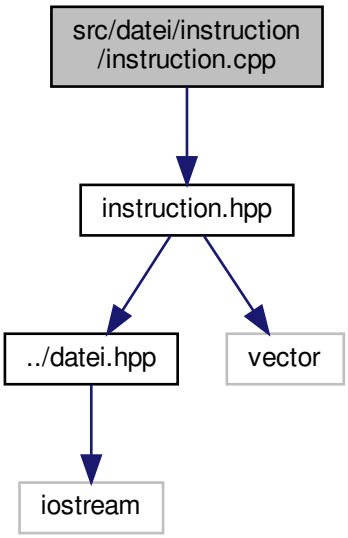

### Data Structures

- class Datei

## 9.8 src/datei/instruction/instruction.cpp File Reference

```
#include "instruction.hpp"
```

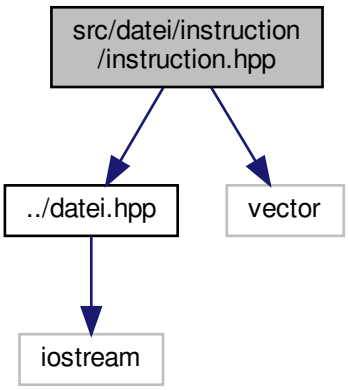
Include dependency graph for instruction.cpp:



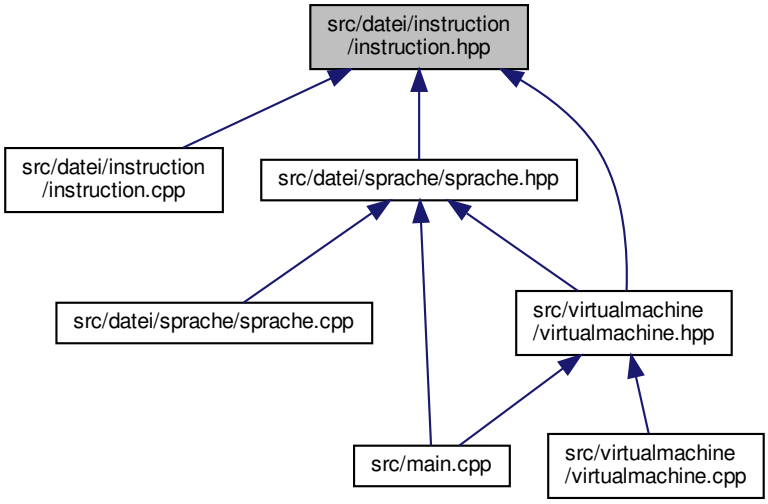

## 9.9 src/datei/instruction/instruction.hpp File Reference

```
#include "../datei.hpp"
#include <vector>
```

Include dependency graph for instruction.hpp:



This graph shows which files directly or indirectly include this file:



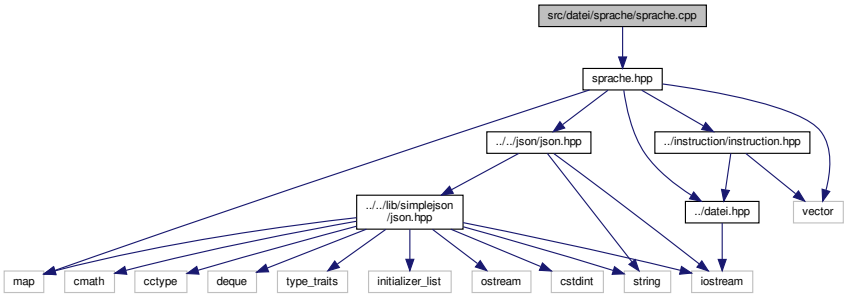

### Data Structures

- class Instruktion
- class SimpleInstruktion
- class ComplexInstruktion

## 9.10 src/datei/sprache/sprache.cpp File Reference

```
#include "sprache.hpp"
```

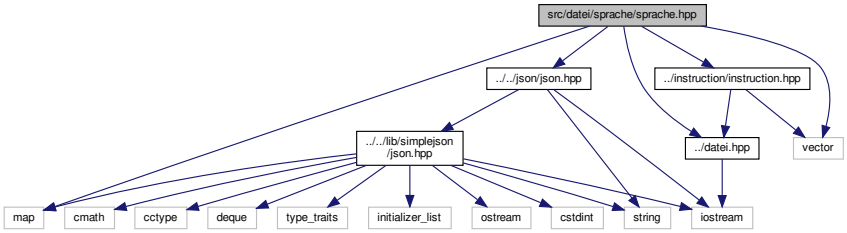
Include dependency graph for sprache.cpp:



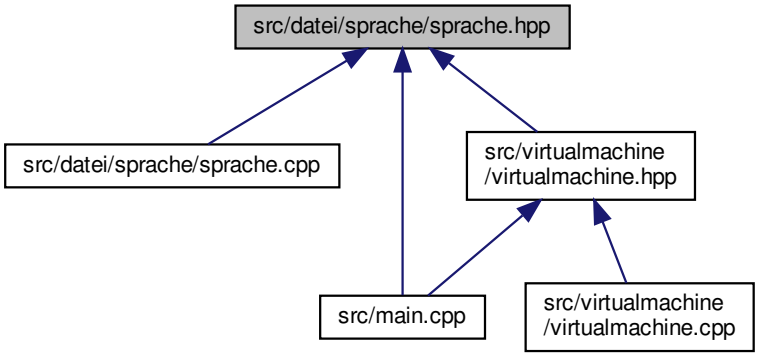
## 9.11 src/datei/sprache/sprache.hpp File Reference

```
#include <map>
#include <vector>
#include "../datei.hpp"
#include "../../json/json.hpp"
#include "../instruction/instruction.hpp"
```

Include dependency graph for sprache.hpp:




This graph shows which files directly or indirectly include this file:




### Data Structures

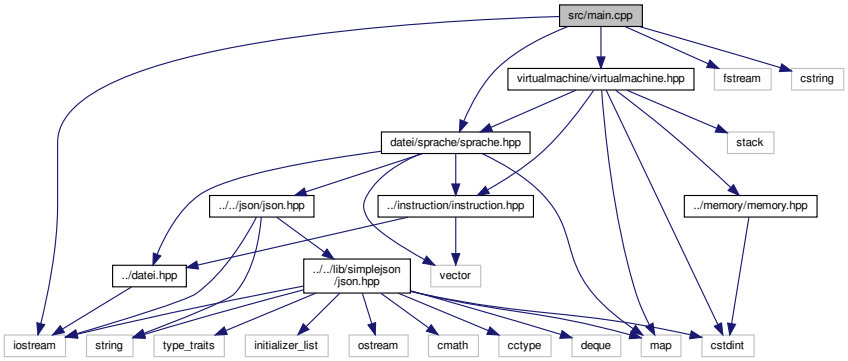
- class Sprache

## 9.12 src/main.cpp File Reference

```
#include <iostream>
#include <fstream>
#include "datei/sprache/sprache.hpp"
#include "virtualmachine/virtualmachine.hpp"
```

```
#include <cstring>
```

Include dependency graph for main.cpp:




## Macros

- #define PRINT_LANGUAGE "-p"
- #define LANGUAGE_FILE "-l"
- #define USE_FILES "-f"

## Functions

- int main (int argc, char **argv)

### 9.12.1 Macro Definition Documentation

#### 9.12.1.1 LANGUAGE_FILE

```
#define LANGUAGE_FILE "-l"
```

Definition at line 9 of file main.cpp.

Referenced by main().

#### 9.12.1.2 PRINT_LANGUAGE

```
#define PRINT_LANGUAGE "-p"
```

Definition at line 8 of file main.cpp.

Referenced by main().

#### 9.12.1.3 USE_FILES

```
#define USE_FILES "-f"
```

Definition at line 10 of file main.cpp.

Referenced by main().

### 9.12.2 Function Documentation

#### 9.12.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Main Funkion des Programmes

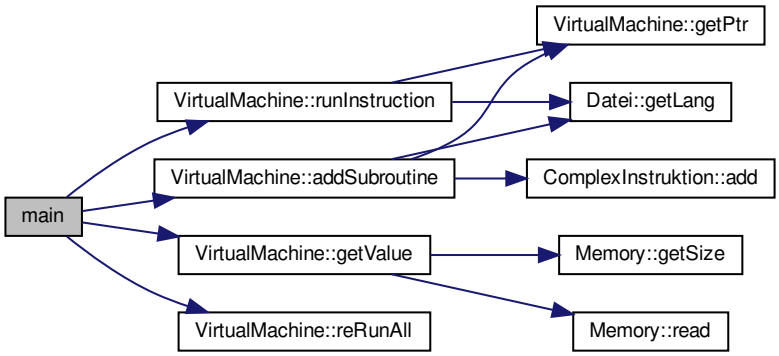
Definition at line 16 of file main.cpp.

References VirtualMachine::addSubroutine(), VirtualMachine::getValue(), LANGUAGE_FILE, PRINT_LANGUAGE, VirtualMachine::reRunAll(), VirtualMachine::runInstruction(), and USE_FILES.

```
17 {
18
19
20
21 std::string sprachendatei = "{  \
22                                                     \n\"Move B to A\":\"mov\",\
23                                                     \n\"Add B to A\":\"add\",\
24                                                     \n\"Substract B from A\":\"sub\",\
25                                                     \n\"Swap the upper and lower 4 bits\":\"swp\",\
26                                                     \n\"Shift left, insert 0\":\"sl0\",\
27                                                     \n\"Shift right, insert 0\":\"sr0\",\
28                                                     \n\"Jump to subroutine\":\"jsr\",\
29                                                     \n\"Push value to stack\":\"push\",\
30                                                     \n\"Pop value from stack\":\"pop\"\
31                                                     \n}";
32 Sprache sprache(sprachendatei, "mylang");
33 VirtualMachine* vm = new VirtualMachine(sprache, 512, 60);
34     for(int i = 1; i<argc;i++){
35         if      (strcmp(argv[i], PRINT_LANGUAGE) == 0){
36             std::ofstream myfile;
37             myfile.open (argv[i+1]);
38             myfile << sprachendatei;
39             myfile.close();
40             return 0;
41         }
42         else if (strcmp(argv[i], LANGUAGE_FILE) == 0){
43             std::ifstream t(argv[++i]);
44             std::string file((std::istreambuf_iterator<char>(t)),
45             std::istreambuf_iterator<char>());
46             sprachendatei = file;
47
48             Sprache sprache(sprachendatei, "mylang");
49             delete vm;
50             vm = new VirtualMachine(sprache);
51
52         }
53         else if (strcmp(argv[i], USE_FILES) == 0){
54             for(i=i+1;i<argc;i++)
55             {
56                 std::ifstream t(argv[i]);
57                 std::string file((std::istreambuf_iterator<char>(t)),
58                 std::istreambuf_iterator<char>());
59                 vm->addSubroutine(file);
```

```
60              }
61          }
62          else{
63              std::cout<<"Unrecognized CLI switch "<<argv[i]<<" , quitting."<<std::endl;
64              return -1;
65          }
66      }
67      std::string line;
68      while(true)
69      {
70          std::getline(std::cin, line);
71          if(line == "quit")
72          {
73              delete vm;
74               return 0;
75          }
76          else if (line == "runAll") vm->reRunAll();
77          else if (line[0] == '(' || line[0] == 'r') std::cout<<"Wert von "<<line<<": 0x"<<std::hex<<(
    unsigned int)vm->getValue(line)<<std::endl;
78          else {
79              try{
80                  vm->runInstruction(line);
81              }
82              catch(char const* e)
83              {
84                  std::cout<<e<<std::endl;
85              }
86          }
87      }
88  }
```

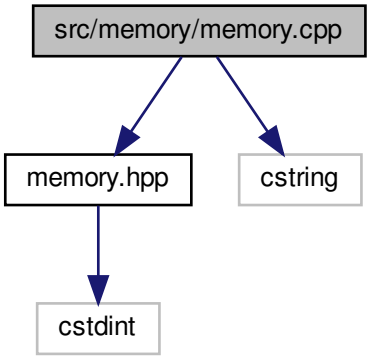
Here is the call graph for this function:




## 9.13 src/memory/memory.cpp File Reference

```
#include "memory.hpp"
#include <cstring>
```

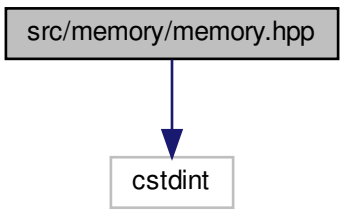
Include dependency graph for memory.cpp:



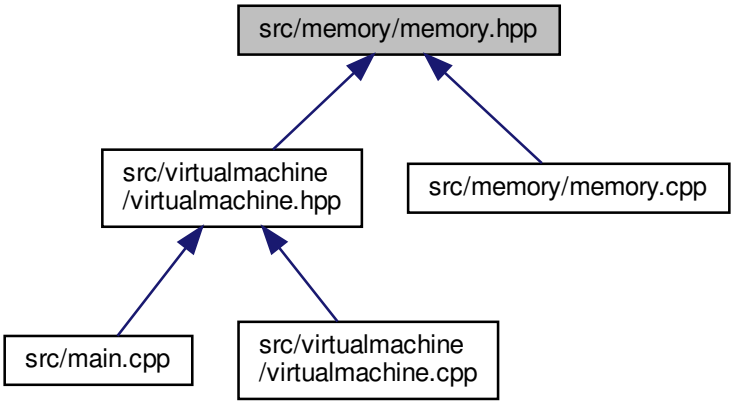

## 9.14 src/memory/memory.hpp File Reference

```
#include <cstdint>
```

Include dependency graph for memory.hpp:



This graph shows which files directly or indirectly include this file:



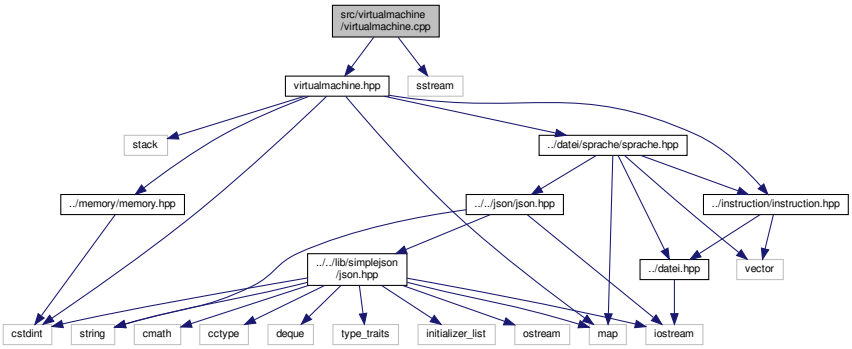

### Data Structures

- class Memory

## 9.15 src/virtualmachine/virtualmachine.cpp File Reference

```
#include "virtualmachine.hpp"
#include <sstream>
```

Include dependency graph for virtualmachine.cpp:




### Functions

- void mov (VirtualMachine &vm, std::string s1, std::string s2)
- void add (VirtualMachine &vm, std::string s1, std::string s2)

- void sub (VirtualMachine &vm, std::string s1, std::string s2)
- void swp (VirtualMachine &vm, std::string s1, std::string s2)
- void sl0 (VirtualMachine &vm, std::string s1, std::string s2)
- void sr0 (VirtualMachine &vm, std::string s1, std::string s2)
- void jsr (VirtualMachine &vm, std::string s1, std::string s2)
- void push (VirtualMachine &vm, std::string s1, std::string s2)
- void pop (VirtualMachine &vm, std::string s1, std::string s2)

## 9.15.1 Function Documentation

### 9.15.1.1 add()

```
void add (
            VirtualMachine & vm,
            std::string s1,
            std::string s2 )
```

Instruktiondefinition.

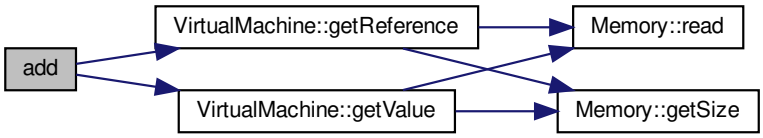
Definition at line 17 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::getValue().

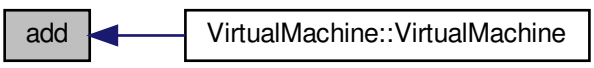
Referenced by VirtualMachine::VirtualMachine().

```
17                                                          {
18     vm.getReference(s1) = vm.getValue(s2) + vm.getValue(s1);
19 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.2 jsr()

```
void jsr (
              VirtualMachine & vm,
              std::string s1,
              std::string s2 )
```

Instruktiondefinition.

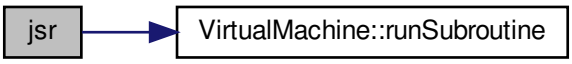
Definition at line 52 of file virtualmachine.cpp.

References VirtualMachine::runSubroutine().

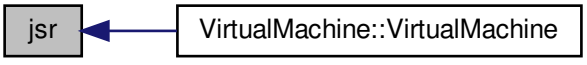
Referenced by VirtualMachine::VirtualMachine().

```
52                                                  {
53     vm.runSubroutine(s1);
54 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.3 mov()

```
void mov (
              VirtualMachine & vm,
              std::string s1,
              std::string s2 )
```

Instruktiondefinition.

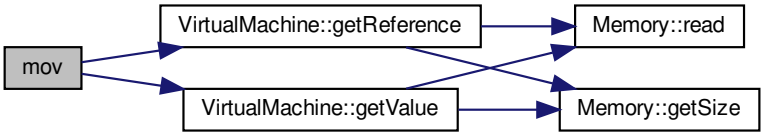
Definition at line 10 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::getValue().

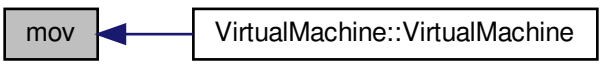
Referenced by VirtualMachine::VirtualMachine().

```
10                                                          {
11     vm.getReference(s1) = vm.getValue(s2);
12 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.4 pop()

```
void pop (
            VirtualMachine & vm,
            std::string s1,
            std::string s2 )
```

Instruktiondefinition.

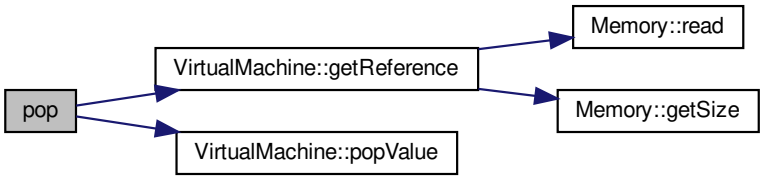
Definition at line 66 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::popValue().

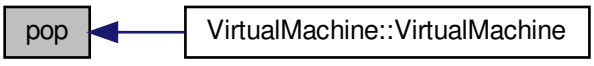
Referenced by VirtualMachine::VirtualMachine().

```
66                                                          {
67     vm.getReference(s1) = vm.popValue();
68 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.5 push()

```
void push (
            VirtualMachine & vm,
            std::string s1,
            std::string s2 )
```

Instruktiondefinition.

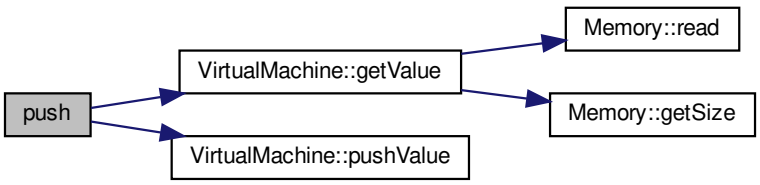
Definition at line 59 of file virtualmachine.cpp.

References VirtualMachine::getValue(), and VirtualMachine::pushValue().

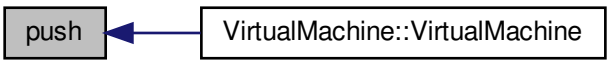
Referenced by VirtualMachine::VirtualMachine().

```
59                                                      {
60     vm.pushValue(vm.getValue(s1));
61 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.6 sl0()

```
void sl0 (
            VirtualMachine & vm,
            std::string s1,
            std::string s2 )
```

Instruktiondefinition.

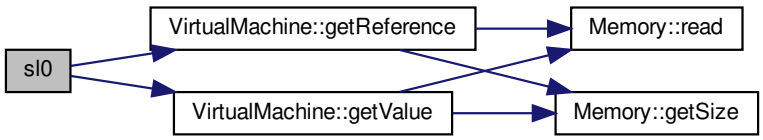
Definition at line 38 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::getValue().

Referenced by VirtualMachine::VirtualMachine().

```
38                                                      {
39     vm.getReference(s1) = vm.getValue(s1) << 1;
40 }
```

Here is the call graph for this function:




Here is the caller graph for this function:

#### 9.15.1.7 sr0()

```
void sr0 (
            VirtualMachine & vm,
            std::string s1,
            std::string s2 )
```

Instruktiondefinition.

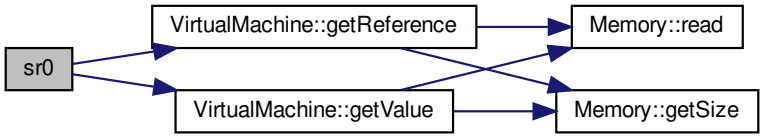
Definition at line 45 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::getValue().

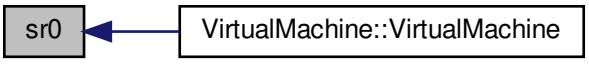
Referenced by VirtualMachine::VirtualMachine().

```
45                                                      {
46     vm.getReference(s1) = vm.getValue(s1) >> 1;
47 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.8 sub()

```
void sub (
              VirtualMachine & vm,
              std::string s1,
              std::string s2 )
```

Instruktiondefinition.

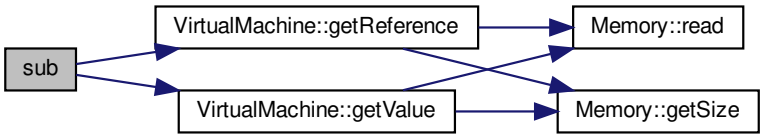
Definition at line 24 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::getValue().

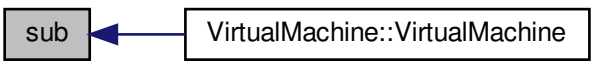
Referenced by VirtualMachine::VirtualMachine().

```
24                                                          {
25     vm.getReference(s1) = vm.getValue(s1) - vm.getValue(s2);
26 }
```

Here is the call graph for this function:




Here is the caller graph for this function:



#### 9.15.1.9 swp()

```
void swp (
             VirtualMachine & vm,
             std::string s1,
             std::string s2 )
```

Instruktiondefinition.

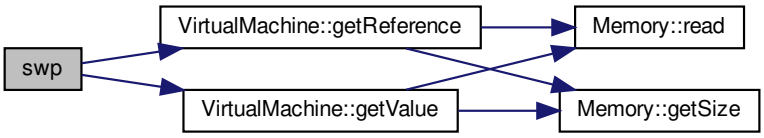
Definition at line 31 of file virtualmachine.cpp.

References VirtualMachine::getReference(), and VirtualMachine::getValue().

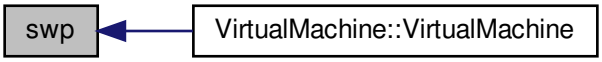
Referenced by VirtualMachine::VirtualMachine().

```
31                                                          {
32     vm.getReference(s1) = (vm.getValue(s1))  >> 4 | (vm.
       getValue(s1) << 4);
33 }
```

Here is the call graph for this function:




Here is the caller graph for this function:

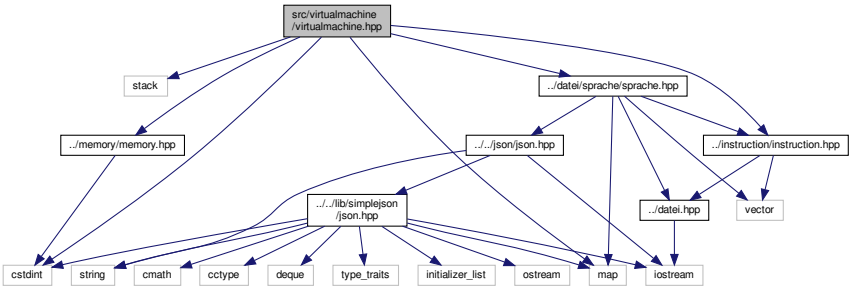



## 9.16 src/virtualmachine/virtualmachine.hpp File Reference

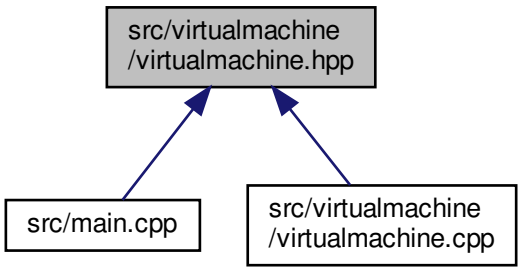
```
#include <stack>
#include <cstdint>
#include <map>
#include "../memory/memory.hpp"
#include "../datei/sprache/sprache.hpp"
```

```
#include "../datei/instruction/instruction.hpp"
```

Include dependency graph for virtualmachine.hpp:




This graph shows which files directly or indirectly include this file:




## Data Structures

- class VirtualMachine

# Index