BinDrv library

1.4

REFERENCE MANUAL

FICO® Xpress Optimization

FICO

©2011–2026 Fair Isaac Corporation. All rights reserved. This documentation is the property of Fair Isaac Corporation ("FICO"). Receipt or possession of this documentation does not convey rights to disclose, reproduce, make derivative works, use, or allow others to use it except solely for internal evaluation purposes to determine whether to purchase a license to the software described in this documentation, or as otherwise set forth in a written software license agreement between you and FICO (or a FICO affiliate). Use of this documentation and the software described in it must conform strictly to the foregoing permitted uses, and no other use is permitted.

The information in this documentation is subject to change without notice. If you find any problems in this documentation, please report them to us in writing. Neither FICO nor its affiliates warrant that this documentation is error-free, nor are there any other warranties with respect to the documentation except as may be provided in the license agreement. FICO and its affiliates specifically disclaim any warranties, express or implied, including, but not limited to, non-infringement, merchantability and fitness for a particular purpose. Portions of this documentation and the software described in it may contain copyright of various authors and may be licensed under certain third-party licenses identified in the software, documentation, or both.

In no event shall FICO or its affiliates be liable to any person for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this documentation or the software described in it, even if FICO or its affiliates have been advised of the possibility of such damage. FICO and its affiliates have no obligation to provide maintenance, support, updates, enhancements, or modifications except as required to licensed users under a license agreement.

FICO is a registered trademark of Fair Isaac Corporation in the United States and may be a registered trademark of Fair Isaac Corporation in other countries. Other product and company names herein may be trademarks of their respective owners.

Patent(s): www.fico.com/en/patents

BinDrv 1.4 (FICO® Xpress 9.9)

Deliverable Version: A

Last Revised: January 2021

Contents

1 Introduction 1

1.1 Generating a data file . . . 1
1.2 Parsing a data file . . . 1
1.3 Building files for Mosel . . . 2

2 Functions of the BinDrv library 3

2.1 Writer . . . 3
bindrv_newwriter . . . 4
bindrv_delete . . . 5
bindrv_putint . . . 6
bindrv_putreal . . . 7
bindrv_putstring . . . 8
bindrv_putbool . . . 9
bindrv_putdata . . . 10
bindrv_putlong . . . 11
bindrv_putctrl . . . 12
2.2 Reader . . . 13
bindrv_newreader . . . 14
bindrv_setalloc . . . 15
bindrv_nexttoken . . . 16
bindrv_getint . . . 17
bindrv_getreal . . . 18
bindrv_getbool . . . 19
bindrv_getstring . . . 20
bindrv_getlong . . . 21
bindrv_getdata . . . 22
bindrv_getctrl . . . 23

Appendix 24

A Contacting FICO 24

FICO Customer Support . . . 24
Documentation . . . 24
FICO Learning . . . 25
Sales and maintenance . . . 25
About FICO . . . 25

Index 26

CHAPTER 1

Introduction

The *BinDrv* library provides the necessary routines to generate and parse data streams using a structured binary and platform independent format. This data format may be employed to generate data files that have to be exchanged between platforms with different byte ordering (or *endianness*). This is also the file format used by the `"bin:"` Mosel I/O driver such, that this library enables an application to create (and read) binary data files to be used by Mosel models.

1.1 Generating a data file

To generate a data file, a *bindrv writer context* has to be allocated with a call to `bindrv_newwriter`. This initialisation routine requires a callback function that is used for outputing data to some stream (*e.g.* use the C function `fwrite` to write to a file opened with the function `fopen`). Then, each data value to be saved to the stream is passed to the function corresponding to its type: this routine builds a token equivalent to the provided object and sends it to the stream. For instance, function `bindrv_putint` will be used to output an integer *etc.*. In order to structure the data flow, special markers may be inserted: the function `bindrv_putctrl` allows to add these control tokens. A control token is characterised by its code (an integer between 0 and 31) such that different types of markers can be used. For instance, control code 1 could indicate the beginning of a list of integers, code 4 the beginning of a list of text strings, code 2 for the end of a list and code 3 could be used to mark the end of the data flow. Using this convention, a file containing a list of integers (*e.g.* 5,6,7) and a list of strings (*e.g.* "a","b") could be produced with the following code:

```
bindrv_putctrl(bctx,1);
bindrv_putint(bctx,5);
bindrv_putint(bctx,6);
bindrv_putint(bctx,7);
bindrv_putctrl(bctx,2);
bindrv_putctrl(bctx,4);
bindrv_putstring(bctx,"a");
bindrv_putstring(bctx,"b");
bindrv_putctrl(bctx,2);
bindrv_putctrl(bctx,3);
```

1.2 Parsing a data file

To parse a data file, a *bindrv reader context* has to be allocated with a call to `bindrv_newreader`. This initialisation routine requires a callback function that is used for inputing data from some stream (*e.g.* use the C function `fread` to read from a file opened with the function `fopen`). The reader procedure has then to enter a loop calling first `bindrv_nexttoken` to get the type of the next token to read and then, depending on this data type, use the function suitable to read and decode this token. For instance if the function returns `BINDRV_TYP_REAL`, a real has to be read and the function `bindrv_getreal` can be used to get its value. If the file structure is known in advance, the data can be

input with a sequence of calls to the decoding functions. For instance the data file of the example of the preceding section could be input with the following code:

```
int a,c;
char *l;

if(bindrv_getctrl(bctx,&c)==0 && (c==1))
{
 while(bindrv_getint(bctx,&a)==0)
  printf("got: %d\n",a);
 if(bindrv_getctrl(bctx,&c)!=0 || (c!=2))
  exit(1);
}
if(bindrv_getctrl(bctx,&c)==0 && (c==4))
{
 while(bindrv_getstring(bctx,&l)==0)
 {
  printf("got: %s\n",l);
  free(l);
 }
 if(bindrv_getctrl(bctx,&c)!=0 || (c!=2))
  exit(2);
}
if(bindrv_getctrl(bctx,&c)!=0 || (c!=3))
 exit(3);
```

As shown above, the application is in charge of releasing string buffers returned by `bindrv_getstring`: this routine allocates the returned memory block using the C function `malloc`. The application may replace this default memory allocator by calling function `bindrv_setalloc`.

1.3 Building files for Mosel

Data files generated for Mosel models must observe the same structure as the usual ascii Mosel format. For instance, consider the following data file in its ascii form:

```
a:123
b:[(1) 10]
```

The corresponding data stored in the binary format can be obtained with the following code:

```
bindrv_putctrl(bctx,BINDRV_CTRL_LABEL);
bindrv_putstring(bctx,"a");
bindrv_putint(bctx,123);
bindrv_putctrl(bctx,BINDRV_CTRL_LABEL);
bindrv_putstring(bctx,"b");
bindrv_putctrl(bctx,BINDRV_CTRL_OPENLST);
bindrv_putctrl(bctx,BINDRV_CTRL_OPENNDX);
bindrv_putint(bctx,1);
bindrv_putctrl(bctx,BINDRV_CTRL_CLOSENDX);
bindrv_putint(bctx,10);
bindrv_putctrl(bctx,BINDRV_CTRL_CLOSELST);
```

Note the use of control tokens to structure the data stream: a dedicated control code corresponds to each of the special characters in the ascii format. For example, the symbol `'['` in the ascii format is represented by the code `BINDRV_CTRL_OPENLST` in the binary format.

CHAPTER 2

Functions of the BinDrv library

2.1 Writer

`bindrv_delete`	Release a BinDrv context.	p. 5
`bindrv_newwriter`	Create a new BinDrv context for writing to an output stream.	p. 4
`bindrv_putbool`	Write a Boolean value to the output stream.	p. 9
`bindrv_putctrl`	Write a control token to the output stream.	p. 12
`bindrv_putdata`	Write a binary data block to the output stream.	p. 10
`bindrv_putint`	Write an integer to the output stream.	p. 6
`bindrv_putlong`	Write a long integer to the output stream.	p. 11
`bindrv_putreal`	Write a real to the output stream.	p. 7
`bindrv_putstring`	Write a text string to the output stream.	p. 8

bindrv_newwriter

Purpose

Create a new BinDrv context for writing to an output stream.

Synopsis

```
s_bindrvctx bindrv_newwriter(size_t (*dowrite)(const void
      *,size_t,size_t,void*), void *rctx);
```

Arguments

`dowrite`	Callback function to write data to the output stream
`rctx`	File descriptor to be passed as the last argument of `dowrite`

Return value

The new context or `NULL` in case of error.

Example

In the following example the file `"bindata` is open using C-function `fopen` and a writer is created based on the resulting file descriptor:

```
f=fopen("bindata", "w");
bdrv=bindrv_newwriter(
       (size_t (*)(const void *,size_t,size_t,void*))fwrite, f);
```

Further information

1. Each context created using this function must be released by a call to `bindrv_delete`.
2. The `dowrite` function is used by the writer whenever it needs to write data. The function receives as input a buffer, its size and, as the last argument, the pointer `rctx`. The third argument is always 1. The return value must be `1` if successful, any other value is interpreted as an error condition.
3. The required `dowrite` routine has the same signature as the C-function `fwrite` such that a BinDrv writer can use as input a file open with the C-function `fopen`.

Related topics

`bindrv_delete`.

bindrv_delete

Purpose

Release a BinDrv context.

Synopsis

```
void bindrv_delete(s_bindrvctx bctx);
```

Argument

`bctx` A BinDrv context

Further information

The same routine is used to release a reader or a writer.

Related topics

`bindrv_newreader`, `bindrv_newwriter`.

bindrv_putint

Purpose

Write an integer to the output stream.

Synopsis

```
int bindrv_putint(s_bindrvctx bctx, int val);
```

Arguments

`bctx` A BinDrv writer context

`val` Integer value to write

Return value

0 if successful, 1 in case of an I/O error

bindrv_putreal

Purpose

Write a real to the output stream.

Synopsis

```
int bindrv_putreal(s_bindrvctx bctx, double val);
```

Arguments

`bctx`	A BinDrv writer context
`val`	Double value to write

Return value

0 if successful, 1 in case of an I/O error

bindrv_putstring

Purpose

Write a text string to the output stream.

Synopsis

```
int bindrv_putstring(s_bindrvctx bctx, const char *val);
```

Arguments

`bctx` A BinDrv writer context

`val` Text string to write

Return value

0 if successful, 1 in case of an I/O error

Further information

The `NULL` pointer is treated as an empty string.

bindrv_putbool

Purpose

Write a Boolean value to the output stream.

Synopsis

```
int bindrv_putbool(s_bindrvctx bctx, char val);
```

Arguments

`bctx` A BinDrv writer context

`val` Boolean value to write

Return value

0 if successful, 1 in case of an I/O error

Further information

The C convention is used: 0 for *false* and any other value for *true*.

bindrv_putdata

Purpose

Write a binary data block to the output stream.

Synopsis

```
int bindrv_putdata(s_bindrvctx bctx, const void *buf, size_t size);
```

Arguments

`bctx`	A BinDrv writer context
`buf`	An array of bytes to write
`size`	Size of the data buffer in bytes

Return value

0 if successful, 1 in case of an I/O error

Further information

If the size is `0` or the input buffer is `NULL` an empty block will be recorded.

bindrv_putlong

Purpose

Write a long integer to the output stream.

Synopsis

```
int bindrv_putlong(s_bindrvctx bctx, BINDRV_LONG val);
```

Arguments

`bctx` A BinDrv writer context

`val` Long integer value (64bit) to write

Return value

0 if successful, 1 in case of an I/O error

Further information

Mosel does not support long integers: a file containing long integers cannot be processed by the `"bin:"` I/O driver.

bindrv_putctrl

Purpose

Write a control token to the output stream.

Synopsis

```
int bindrv_putctrl(s_bindrvctx bctx, int val);
```

Arguments

`bctx` A BinDrv reader context

`val` Control code to write (value between 0 and 31)

Return value

0 if successful, 1 in case of an I/O error

Further information

1. A control code is represented by a 5bit integer (values 0 to 31) the interpretation of which is application-specific: the `bindrv` library does not use these control tokens. They are usually used as markers to structure the data flow: it is up to the user to define appropriate conventions according to his needs.

2. When generating a data stream to be parsed by a Mosel model, the following control codes must be used — they correspond to the control characters employed in the standard Mosel ascii file format:

   `BINDRV_CTRL_SKIP` Skip entry (symbol `'*'` in ascii format)

   `BINDRV_CTRL_LABEL` Start a new record: this control token is always followed by a string identifying the label (corresponds to `'label:'` in ascii format)

   `BINDRV_CTRL_OPENLST` Begin list of entries (symbol `'['` in ascii format)

   `BINDRV_CTRL_CLOSELST` End list of entries (symbol `']'` in ascii format)

   `BINDRV_CTRL_OPENNDX` Begin list of array indices (symbol `'('` in ascii format)

   `BINDRV_CTRL_CLOSENDX` End list of array indices (symbol `')'` in ascii format)

   `BINDRV_CTRL_NIL` NIL entry (symbol `'?'` in ascii format)

2.2 Reader

`bindrv_getbool`	Get the value of a Boolean token.	p. 19
`bindrv_getctrl`	Get the code of a control token.	p. 23
`bindrv_getdata`	Get the value of a data block token.	p. 22
`bindrv_getint`	Get the value of an integer token.	p. 17
`bindrv_getlong`	Get the value of a long integer token.	p. 21
`bindrv_getreal`	Get the value of a real token.	p. 18
`bindrv_getstring`	Get the value of a text string token.	p. 20
`bindrv_newreader`	Create a new BinDrv context for parsing an input stream.	p. 14
`bindrv_nexttoken`	Get the type of the next token to read.	p. 16
`bindrv_setalloc`	Define a memory allocator for string buffers.	p. 15

bindrv_newreader

Purpose

Create a new BinDrv context for parsing an input stream.

Synopsis

```
s_bindrvctx bindrv_newreader(size_t (*doread)(void *,size_t,size_t,void*),
      void *rctx);
```

Arguments

`doread`	Callback function to retrieve data from the input stream
`rctx`	File descriptor to be passed as the last argument of `doread`

Return value

The new context or `NULL` in case of error.

Example

In the following example the file `"bindata` is opened using the C function `fopen` and a reader is created based on the resulting file descriptor:

```
f=fopen("bindata", "r");
bdrv=bindrv_newreader((size_t (*)(void *,size_t,size_t,void*))fread, f);
```

Further information

1. Each context created using this function must be released by a call to `bindrv_delete`.
2. The `doread` routine is used by the reader whenever it requires further data to process. The function receives as input a buffer (where read data has to be copied), its size and, as the last argument, the pointer `rctx`. The third argument of `doread` is always 1. The return value must be `1` if successful (*i.e.* the requested amount of data has been read), any other value is interpreted as an error condition.
3. The required `doread` function has the same signature as the C function `fread` such that a BinDrv reader can use as input a file opened with the C function `fopen`.

Related topics

`bindrv_delete`.

bindrv_setalloc

Purpose

Define a memory allocator for string buffers.

Synopsis

```
void bindrv_setalloc(s_bindrvctx bctx, void* (*memalloc)(size_t,void*),
      void* mctx);
```

Arguments

`bctx`	A BinDrv reader context
`memalloc`	A memory allocator or `NULL`
`mctx`	Context to be passed as the last argument of `mamalloc`

Further information

1. When reading a string, the function `bindrv_getstring` allocates a buffer that it returns. By default the C function `malloc` is used for this memory allocation. This routine allows to set an alternative function for this task: the provided function takes as argument the size of the buffer to allocate as well as some context pointer `mctx`.

2. Using `NULL` as the function pointer restores the default memory allocator, namely the C function `malloc`.

Related topics

`bindrv_getstring`.

bindrv_nexttoken

Purpose

Get the type of the next token to read.

Synopsis

```
int bindrv_nexttoken(s_bindrvctx bctx);
```

Argument

`bctx` A BinDrv reader context

Return value

Type code for the next token or a negative value in case of a read error (end of stream). Possible values are:

`BINDRV_TYP_INT` An integer

`BINDRV_TYP_DATA` A binary data block

`BINDRV_TYP_REAL` A real (as a double)

`BINDRV_TYP_STR` A text string

`BINDRV_TYP_BOOL` A Boolean

`BINDRV_TYP_CTRL` A control token

`BINDRV_TYP_LONG` A long integer (64bit)

Further information

This routine does not return the token itself but indicates which function to use for reading it. For instance, if the return value is `BINDRV_TYP_INT`, the function `bindrv_getint` has to be called to retrieve an integer value.

Related topics

`bindrv_getint`, `bindrv_getdata`, `bindrv_getreal`, `bindrv_getstring`, `bindrv_getbool`, `bindrv_getctrl`, `bindrv_getlong`.

bindrv_getint

Purpose

Get the value of an integer token.

Synopsis

```
int bindrv_getint(s_bindrvctx bctx, int *val);
```

Arguments

`bctx` A BinDrv reader context

`val` Pointer to return the integer value

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Related topics

`bindrv_nexttoken`.

bindrv_getreal

Purpose

Get the value of a real token.

Synopsis

```
int bindrv_getreal(s_bindrvctx bctx, double *val);
```

Arguments

`bctx`	A BinDrv reader context
`val`	Pointer to return the double value

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Related topics

`bindrv_nexttoken`.

bindrv_getbool

Purpose

Get the value of a Boolean token.

Synopsis

```
int bindrv_getbool(s_bindrvctx bctx, char *val);
```

Arguments

`bctx`	A BinDrv reader context
`val`	Pointer to return the Boolean value (as a `char`)

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Related topics

`bindrv_nexttoken`.

bindrv_getstring

Purpose

Get the value of a text string token.

Synopsis

```
int bindrv_getstring(s_bindrvctx bctx,char **val);
```

Arguments

`bctx`	A BinDrv reader context
`val`	Pointer to return a reference to the string

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Further information

The returned buffer is allocated using the C function `malloc`. To replace this default memory allocator, (*e.g.* by some application specific memory management routine), use `bindrv_setalloc`.

Related topics

`bindrv_nexttoken`.

bindrv_getlong

Purpose

Get the value of a long integer token.

Synopsis

```
int bindrv_getlong(s_bindrvctx bctx,BINDRV_LONG *val);
```

Arguments

`bctx`	A BinDrv reader context
`val`	Pointer to return the long integer value (64bit)

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Further information

Mosel does not support long integers: this type of token will never be returned from a file generated by Mosel.

Related topics

`bindrv_nexttoken`.

bindrv_getdata

Purpose

Get the value of a data block token.

Synopsis

```
int bindrv_getdata(s_bindrvctx bctx,void **val, size_t *size);
```

Arguments

`bctx`	A BinDrv reader context
`val`	Pointer to return a reference to the data buffer
`size`	Pointer to return the size of the data buffer

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Further information

1. The returned buffer is allocated using the C function `malloc`. To replace this default memory allocator, (*e.g.* by some application specific memory management routine), use `bindrv_setalloc`.
2. The buffer reference will be `NULL` and the size will be `0` when an empty block is encountered.

Related topics

`bindrv_nexttoken`.

bindrv_getctrl

Purpose

Get the code of a control token.

Synopsis

```
int bindrv_getctrl(s_bindrvctx bctx,int *val);
```

Arguments

`bctx` A BinDrv reader context

`val` Pointer to return the control code

Return value

0 if successful, a negative value in case of error or the type code of the token (positive value) if it is not of the expected type.

Further information

1. A control code is represented by a 5bit integer (values 0 to 31) the interpretation of which is application-specific: the `bindrv` library does not use these control tokens. They are usually used as markers to structure the data flow: it is up to the user to define appropriate conventions according to his needs.

2. When the data stream is coming from Mosel, the following control codes may be used — they correspond to the control characters employed in the standard Mosel ascii file format:

   `BINDRV_CTRL_SKIP` Skip entry (symbol `'*'` in ascii format)

   `BINDRV_CTRL_LABEL` Start a new record: this control token is always followed by a string identifying the label (corresponds to `'label:'` in ascii format)

   `BINDRV_CTRL_OPENLST` Begin list of entries (symbol `'['` in ascii format)

   `BINDRV_CTRL_CLOSELST` End list of entries (symbol `']'` in ascii format)

   `BINDRV_CTRL_OPENNDX` Begin list of array indices (symbol `'('` in ascii format)

   `BINDRV_CTRL_CLOSENDX` End list of array indices (symbol `')'` in ascii format)

   `BINDRV_CTRL_NIL` NIL entry (symbol `'?'` in ascii format)

Related topics

`bindrv_nexttoken`.

Appendix A

Contacting FICO

FICO provides clients with support and services for all our products.

FICO Customer Support

FICO Customer Support offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have an active maintenance contract.

The FICO Customer Self-Service Portal (support.fico.com) is a secure web portal that allows users to open, review, and update their support cases; manage their organization's portal users; find solutions to common problems in the FICO Knowledge Base; and view the availability of their cloud applications 24 hours a day, 7 days a week.

You can find support contact information and a link to the FICO Customer Self-Service Portal (online support) on the Product Support home page (www.fico.com/en/product-support).

Please include *'Xpress'* in the subject line of your support queries.

Documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide.

If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com. Please include your contact information (name, company, email address, and optionally, your phone number) so we may reach you if we have questions.

FICO Learning

FICO Learning is the principal provider of product training for our clients and partners. FICO Learning offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support.

For additional information, visit the FICO Learning home page at www.fico.com/en/product-training or email producteducation@fico.com.

Sales and maintenance

If you need information on other Xpress Optimization products, or you need to discuss maintenance contracts or other sales-related items, contact FICO by:

- Phone: +1 (408) 535-1500 or +44 207 940 8718
- Web: www.fico.com/optimization and use the available contact forms

About FICO

FICO (NYSE:FICO) is a leading analytics software company, helping businesses in 90+ countries make better decisions that drive higher levels of growth, profitability, and customer satisfaction. Learn more at www.fico.com or contact us at www.fico.com/en/contact-us.

Index

B
bindrv writer context, 1
`bindrv_delete`, 5
`bindrv_getbool`, 19
`bindrv_getctrl`, 23
`bindrv_getdata`, 22
`bindrv_getint`, 17
`bindrv_getlong`, 21
`bindrv_getreal`, 18
`bindrv_getstring`, 20
`bindrv_newreader`, 14
`bindrv_newwriter`, 4
`bindrv_nexttoken`, 16
`bindrv_putbool`, 9
`bindrv_putctrl`, 12
`bindrv_putdata`, 10
`bindrv_putint`, 6
`bindrv_putlong`, 11
`bindrv_putreal`, 7
`bindrv_putstring`, 8
`bindrv_setalloc`, 15
`BINDRV_CTRL_CLOSELST`, 12, 23
`BINDRV_CTRL_CLOSENDX`, 12, 23
`BINDRV_CTRL_LABEL`, 12, 23
`BINDRV_CTRL_NIL`, 12, 23
`BINDRV_CTRL_OPENLST`, 12, 23
`BINDRV_CTRL_OPENNDX`, 12, 23
`BINDRV_CTRL_SKIP`, 12, 23
`BINDRV_TYP_BOOL`, 16
`BINDRV_TYP_CTRL`, 16
`BINDRV_TYP_DATA`, 16
`BINDRV_TYP_INT`, 16
`BINDRV_TYP_LONG`, 16
`BINDRV_TYP_REAL`, 16
`BINDRV_TYP_STR`, 16
Boolean
    read, 19
    write, 9

C
context
    input stream, 14
    output stream, 4
    release, 5
control token, 2
    read, 23
    write, 12

I
input parsing, 14
input stream
    create context, 14
integer
    read, 17
    write, 6

L
long integer
    read, 21
    write, 11

M
memory allocator, 15

O
output stream
    create context, 4

R
read
    Boolean, 19
    control token, 23
    integer, 17
    long integer, 21
    real, 18
    string, 20
    token type, 16
real
    read, 18
    write, 7

S
string
    memory allocator, 15
    read, 20
    write, 8

T
token type
    read, 16
token type code, 16

W
write
    Boolean, 9
    control token, 12
    integer, 6
    long integer, 11
    real, 7
    string, 8
writer context, 1