

# Boost.Spirit 2.5.4 Reference Card

## Primitive Parsers

attr(a)	Attribute	A
eoi	End of Input	<i>unused</i>
eol	End of Line	<i>unused</i>
eps	Epsilon	<i>unused</i>
eps(p)		<i>unused</i>
symbols<Ch,T,Lookup>	Symbol Table	T

## Unary Parsers

&a	And Predicate	<i>unused</i>
!a	Not Predicate	<i>unused</i>
*a	Zero or More	vector<A>
*u		<i>unused</i>
-a	Optional	optional<A>
-u		<i>unused</i>
+a	One or More	vector<A>
+u		<i>unused</i>
attr_cast<T>(p)	Attribute Cast	T

## Binary Parsers

a - b	Difference	A
u - b		<i>unused</i>
a % b	Separated List	vector<A>
u % b		<i>unused</i>

## N-ary Parsers

a   b	Alternative	variant<A, B>
a   u		optional<A>
a   b   u		optional<variant<A,B>>
u   b		optional<B>
u   u		<i>unused</i>
a   a		A
a > b	Expect	tuple<A,B>
a > u		A
u > b		B
u > u		<i>unused</i>
a > vA		vector<A>
vA > a		vector<A>
vA > vA		vector<A>
a ^ b	Permute	tuple<optional<A>,optional<B>>
a ^ u		optional<A>
u ^ b		optional<B>
u ^ u		<i>unused</i>
a >> b	Sequence	tuple<A,B>
a >> u		A
u >> b		B
u >> u		<i>unused</i>
a >> a		vector<A>
a >> vA		vector<A>
vA >> a		vector<A>
vA >> vA		vector<A>
a    b	Sequence Or	tuple<optional<A>,optional<B>>
a    u		optional<A>
u    a		optional<A>
u    u		<i>unused</i>
a    a		vector<optional<A>>

## Nonterminal Parsers

rule<It,RT(A1,...,An),Skip,Loc>	Rule	RT
rule<It>		<i>unused</i>
rule<It,Skip>		<i>unused</i>
rule<It,Loc>		<i>unused</i>
rule<It,Skip,Loc>		<i>unused</i>
grammar<It,RT(A1,...,An),Skip,Loc>	Grammar	RT
grammar<It>		<i>unused</i>
grammar<It,Skip>		<i>unused</i>
grammar<It,Loc>		<i>unused</i>
grammar<It,Skip,Loc>		<i>unused</i>

## Parser Directives

as<T>() [a]	Atomic Assignment	T
expect [a]	Expectation	A
expect [u]		<i>unused</i>
hold [a]	Hold Attribute	A
hold [u]		<i>unused</i>
lexeme [a]	Lexeme	A
lexeme [u]		<i>unused</i>
matches [a]	Matches	bool
no_case [a]	Case Insensitive	A
no_case [u]		<i>unused</i>
no_skip [a]	No Skipping	A
no_skip [u]		<i>unused</i>
omit [a]	Omit Attribute	<i>unused</i>
raw [a]	Raw Iterators	iterator_range<It>
raw [u]		<i>unused</i>
repeat [a]	Repeat	vector<A>
repeat [u]		<i>unused</i>
repeat (n) [a]		vector<A>
repeat (n) [u]		<i>unused</i>
repeat (min,max) [a]		vector<A>
repeat (min,max) [u]		<i>unused</i>
repeat (min,inf) [a]		vector<A>
repeat (min,inf) [u]		<i>unused</i>
skip [a]	Skip Whitespace	A
skip [u]		<i>unused</i>
skip (p) [a]		A
skip (p) [u]		<i>unused</i>

## Semantic Actions

p[fa]	Apply Semantic Action	A
p[ <i>phoenix lambda</i> ]		A
template<typename Attrib>		
void fa(Attrib& attr);		
template<typename Attrib, typename Context>		
void fa(Attrib& attr, Context& context);		
template<typename Attrib, typename Context>		
void fa(Attrib& attr, Context& context, bool& pass);		

## Phoenix Placeholders

_1, _2, ..., _N	Nth Attribute of p
_val	Enclosing rule's synthesized attribute
_r1, _r2, ..., _rN	Enclosing rule's Nth inherited attribute.
_a, _b, ..., _j	Enclosing rule's local variables.
_pass	Assign false to _pass to force failure.

## Iterator Parser API

bool parse<It, Exp>( It& first, It last, Exp const& expr);	
bool parse<It, Exp, A1, ..., An>( It& first, It last, Exp const& expr, A1& a1, ..., An& an);	
bool phrase_parse<It, Exp, Skipper>( It& first, It last, Exp const& expr, Skipper const& skipper, skip_flag post_skip = postskip);	
bool phrase_parse<It, Exp, Skipper, A1, ..., An>( It& first, It last, Exp const& expr, Skipper const& skipper, A1& a1, ..., An& an);	
bool phrase_parse<It, Exp, Skipper, A1, ..., An>( It& first, It last, Exp const& expr, Skipper const& skipper, skip_flag post_skip, A1& a1, ..., An& an);	

## Stream Parser API

<i>unspecified</i> match<Exp>(Exp const& expr);	
<i>unspecified</i> match<Exp, A1, ..., An>( Exp const& expr, A1& a1, ..., An& an);	
<i>unspecified</i> phrase_match<Exp, Skipper>( Exp const& expr, Skipper const& skipper, skip_flag post_skip = postskip);	
<i>unspecified</i> phrase_match<Exp, Skipper, A1, ..., An>( Exp const& expr, Skipper const& skipper, skip_flag post_skip, A1& a1, ..., An& an);	

Binary Value Parsers

byte_	Native Byte	uint_least8_t
byte_(b)		<i>unused</i>
word	Native Word	uint_least16_t
word(w)		<i>unused</i>
dword	Native Double Word	uint_least32_t
dword(dw)		<i>unused</i>
qword	Native Quad Word	uint_least64_t
qword(qw)		<i>unused</i>
bin_float	Native Float	float
bin_float(f)		<i>unused</i>
bin_double	Native Double	double
bin_double(d)		<i>unused</i>
little_item	Little Endian <i>item</i>	as above
little_item(w)		<i>unused</i>
big_item	Big Endian <i>item</i>	as above
big_item(w)		<i>unused</i>

Character Encodings

ascii	7-bit ASCII
iso8859_1	ISO 8859-1
standard	Using <cctype>
standard_wide	Using <cwctype>

Character Parsers

c	Character Literal	<i>unused</i>
lit(c)		<i>unused</i>
ns::char_	Any Character	ns::char_type
ns::char_(c)	Character Value	ns::char_type
ns::char_(f,l)	Character Range	ns::char_type
ns::char_(str)	Any Character in String	ns::char_type
~cp	Characters not in cp	Attribute of cp

Character Class Parsers

ns::alnum	Letters or Digits	ns::char_type
ns::alpha	Alphabetic	ns::char_type
ns::blank	Spaces or Tabs	ns::char_type
ns::cntrl	Control Characters	ns::char_type
ns::digit	Numeric Digits	ns::char_type
ns::graph	Non-space Printing Characters	ns::char_type
ns::lower	Lower Case Letters	ns::char_type
ns::print	Printing Characters	ns::char_type
ns::punct	Punctuation	ns::char_type
ns::space	White Space	ns::char_type
ns::upper	Upper Case Letters	ns::char_type
ns::xdigit	Hexadecimal Digits	ns::char_type

String Parsers

str	String Literal	<i>unused</i>
lit(str)		<i>unused</i>
ns::string("str")	String	basic_string<char>
ns::string(L"str")		basic_string<wchar_t>

Unsigned Integer Parsers

lit(num)	Integer Literal	<i>unused</i>
ushort_	Short	unsigned short
ushort_(num)	Short Value	unsigned short
uint_	Integer	unsigned int
uint_(num)	Integer Value	unsigned int
ulong_	Long	unsigned long
ulong_(num)	Long Value	unsigned long
ulong_long	Long Long	unsigned long long
ulong_long(num)	Long Long Value	unsigned long long
bin	Binary Integer	unsigned int
bin(num)	Binary Integer Value	unsigned int
oct	Octal Integer	unsigned int
oct(num)	Octal Integer Value	unsigned int
hex	Hexadecimal Integer	unsigned int
hex(num)	Hex Integer Value	unsigned int

Generalized Unsigned Integer Parser

uint_parser<T,Radix,MinDigits,MaxDigits>	T
uint_parser<T,Radix,MinDigits,MaxDigits>(num)	T

Signed Integer Parsers

lit(num)	Integer Literal	<i>unused</i>
short_	Short	short
short_(num)	Short Value	short
int_	Integer	int
int_(num)	Integer Value	int
long_	Long	long
long_(num)	Long Value	long
long_long	Long Long	long long
long_long(num)	Long Long Value	long long

Generalized Signed Integer Parser

int_parser<T,Radix,MinDigits,MaxDigits>	T
int_parser<T,Radix,MinDigits,MaxDigits>(num)	T

Real Number Parsers

lit(num)	Real Number Literal	<i>unused</i>
float_	Float	float
float_(num)	Float Value	float
double_	Double	double
double_(num)	Double Value	double
long_double	Long Double	long double
long_double(num)	Long Double Value	long double

Generalized Real Number Parser

real_parser<T,RealPolicies>	T
real_parser<T,RealPolicies>(num)	T

Boolean Parsers

lit(boolean)	Boolean Literal	<i>unused</i>
false_	Match “false”	bool
true_	Match “true”	bool
bool_	Boolean	bool
bool_(boolean)	Boolean Value	bool

Generalized Boolean Parser

bool_parser<T,BoolPolicies>	T
bool_parser<T,BoolPolicies>(boolean)	T