```
record { record-properties } ( fieldname_1 : fieldtype { field-properties };
                               ...
                               fieldname_n : fieldtype { field-properties };)
```

General Operator Record Schema Syntax

The syntax in this section applies to all Orchestrate operators except for the elements enclosed in curly braces which apply only to the **import** and **export** operators. These exceptions are noted in the subsections below. See the sections "Import/Export Operator Record Schema Syntax" and "Generator Operator Field Properties Syntax" for syntax specific to those operators.

Field Names

A *fieldname* may include single-byte and unicode characters. Its character-set encoding is determined by the **-input_charset** option. It can contain only alphanumeric characters, as defined by the Unicode standard, and underscore characters. It must start with a letter or an underscore character. It is case insensitive and can be of any length.

Field Data Types

Field Data Type	Forms	Explanation
integer	int8, int16, int32, int64 uint8, uint16, uint32, uint64	1, 2, 4, and 8-byte signed integers 1, 2, 4, and 8-byte unsigned integers
floating point	sfloat dfloat	single-precision floating-point value double-precision floating-point value
string for processing single-byte character data (UTF-8)	string string[max=*n_code_point_units*] string[*n_code_point_units*] string[..., padchar = *ASCII_value* | *'ASCII_char'* | null]	variable-length string variable-length with upper bound fixed-length string optional padding character specification A *codepoint* is always equal to a byte.
ustring for processing multi-byte unicode character data (UTF-16)	ustring ustring[max=*n_code_point_units*] ustring[*n_code_point_units*] ustring[..., padchar = *unicode_value* | *'unicode_char'* | null]	variable-length unicode string variable-length unicode string with upper bound fixed-length unicode string optional unicode padding character specification

Field Data Type	Forms	Explanation
decimal	decimal[*precision*] decimal[*precision, scale*]	decimal value (*scale* defaults to 0) decimal value *precision:* 1 - 255 digits 0 <= *scale* < *precision*
date / **time**	date time time[*microseconds*] timestamp timestamp[*microseconds*]	date time with second resolution time with microsecond resolution date / time with second resolution date / time with microsecond resolution There is currently no support for unicode character data for these field types.
raw	raw raw[max=*n*] raw[*n*] raw[..., align=*n*]	variable-length variable-length with upper bound fixed length align on *n*-byte boundary where *n* = 1, 2, or 4 (optional)

Note: Vector fields cannot be tagged, and tagged and subrecord fields are not nullable.

Vector Fields

Vector Field Syntax	Explanation
fieldname[*length*] : *fieldtype*	fixed-length vector
fieldname[] : *fieldtype* { vector_prefix=*n* }	variable length vector; use zero-based indexing to reference an element The vector_prefix syntax, where *n* = 1, 2, or 4, can only be specified for the **import** and **export** operators.
length-field : *uint32* { link }; *fieldname*[] : *fieldtype* { reference=*length-field* };	The link and reference syntax, where the length of the vector is given by the value of the attached *length-field*, can only be specified for the **import** and **export** operators.

Nullable Fields

Nullable Field Syntax	Explanation
fieldname : nullable *fieldtype*	nullable field
fieldname[] : *fieldtype* { vector_prefix=*n* }	vector of nullable values The vector_prefix syntax, where *n* = 1, 2, or 4, can only be specified for the **import** and **export** operators.

Subrecord Fields

Subrecord Field Syntax	Explanation
fieldname : subrec (*subfieldname1*:*fieldtype*; . . . *subfieldnamen*:*fieldtype*;);	Use dot addressing to reference a subrecord: *fieldname.subfieldname* A subrec has its own naming scope. For example, *fieldname* and *subfieldname1* can have the same name. A subrecord field can be placed inside a tagged field.
fieldname[*n*] : subrec (*subfieldname1*:*fieldtype*; . . . *subfieldnamen*:*fieldtype*;);	A vector of subrecords. Use zero-based indexing to reference a vector element and dot addressing to reference a subrecord: *fieldname*[*n*].*subfieldname*
fieldname : subrec (*subfieldname1*:*fieldtype*; . . . *subfieldname2*:subrec (*subfieldname2a*:*fieldtype*; . . . *subfieldname2b*:*fieldtype*;););	Nested subrecords to any depth. To reference a subrecord, use dot addressing. For example: *fieldname.subfieldname2.subfieldname2b*

Tagged Fields

Tagged Field Syntax	Explanation
fieldname : tagged (*tagfield1*:*fieldtype*; . . . *tagfieldn*:*fieldtype*;);	The value of *fieldname* can be the data type of any one of the *tagfields.* A *tagfield* can be any type except tagged or subrec. Use dot addressing to reference a tagged field: *fieldname.tagfield.* A tagged field cannot be a vector. A subrecord field can be placed inside a tagged field.
link-fieldname : uint32 { link } ; *tag-fieldname* : tagged { reference = *link-fieldname* } (*fieldname* 0: *fieldtype* { tagcase = 0 }; *fieldname1* : *fieldtype* { tagcase = 1 }; . . . *fieldnamen* : *fieldtype* { tagcase = *n* };);	The link and reference syntax, where the length of the vector is given by the value of the attached *length-field*, can only be specified for the **import** and **export** operators. The attached *link-fieldname* supplies a tagcase value for *tag-fieldname.* For example, if the value of *link-field* is 1, the data type for *tag-fieldname* is the data type of *fieldname1*.

Import/Export Operator Record Schema Syntax

In record schemas used for the import or export of data, you can use the record- and field-level properties to specify how the data is read or written.

The default schema format for input specifies that each record is a single newline-delimited string:

```
record { text, record_delim='\n' } ( rec: string; )
```

The default schema for export specifies that the fields are space-separated, with the record terminated by a newline character:

```
record { text, delim=' ', record_delim='\n' } ()
```

Import/Export Record Properties

Import/Export Record Properties	Explanation
Intact Properties	

This property lets you improve performance by limiting the amount of processing done on each record of the data. Only fields explicitly specified by the schema are processed. The example below contains 80 characters of data, a character newline delimiter, and references only two fields:
`record { intact, record_length = 81, record_delim='\n' }`
`( int : int32 { position=12, delim=",", text };`
`str : string[20] { position=40, delim=none };`
`)`	

intact	intact record, with no import field checking (faster)
intact, check_intact	the same as above, but enable import field checking (slower)
intact=*recordname*	optionally specify "fieldname" for the entire record
Length Properties	
record_delim	delimited by newline characters, the default
record_delim = 'x' | null	delimited by a single-byte or unicode character or by null (0x00)
record_delim_string = "*string"*	delimited by the specified single-byte character string or unicode string
record_prefix = *n*	*n*-byte length prefix
record_length = *length*	fixed-length records; can be included with each of the other length formats
record_length = fixed	field lengths determine the record length
Format Properties	
record_format={ type=implicit }	record length is determined from field lengths
record_format={type=varying, format = V | VB | VS | VBS	block / spanned formats

Import/Export Record Properties	Explanation
Record-Level Field Properties These properties control the processing of the fields in the record. An example is a record with three comma-separated fields. The last field is delimited by the end of the record: record {delim=',', final_delim=end } (a: int32; b:int32; c:int32;)	
fill = *bytevalue*	on export, fill the gaps between positioned fields with the byte value; ignored on import
final_delim = ws | end | none | null | 'x'	unique deliminter for the last field; precedes record_delim, if any. The options are white space, end-of-record, no delimiter (use field length), null (0x00), or a single-byte or unicode character, respectively. The default is end-of-record.
final_delim_string = "string"	single-byte string or unicode string delimiter for the last field

Import/Export Field Properties

R: A field property preceded by an R can also be used as a record property to specify a default property for all the fields in a record. A field-level property overrides a record-level property.

General Import/Export Field Properties

These properties apply to all field types, except where noted.

	General Import/Export Field Property	Explanation
R	binary	binary format for data values Binary format cannot be used with string or ustring fields.
R	text , ascii | ebcdic , import_ebcdic_as_ascii , export_ebcdic_as_ascii	text representation of values optionally specify character set optionally convert character set on import optionally convert character set on export Text format cannot be used with raw fields or with ustring fields. The charset property below offers similar functionality for ustring fields.
R	charset = *charset*	The specified charset applies to the ustring field. When used at the record level, the charset applies to all ustring fields in the record.
R	prefix = *n*	*n*-byte length prefix where *n* = 1, 2, or 4
R	delim_string = "string"	a single-byte string or unicode string delimiter for the field

	General Import/Export Field Property	Explanation
R	delim = ws | end | none |null |'*x*'	delimiter for field ws = whitespace which is the default end = end of record none = no delimiter (use field length) null = 0x00 '*x*' = specified single-byte or unicode character
R	quote = single | double | '*x*'	the field is enclosed by single-byte or unicode quote characters Quotes are removed on import, and added on export.
R	print_field	print each imported field value; intended for debugging
	default = value	default value to be used if field import or export fails; may contain unicode characters for ustring fields
	drop	drop field on import; ignored on export
	generate	set field to default value on export; ignored on import
	position = *nbytes* | fieldname	*nbytes* is the offset of the field in the record where the first byte = 0; fieldname uses the starting offset of the field
	skip = *nbytes*	skip *nbytes* after preceding field; can be negative to skip backward
	link reference = *linkfieldname*	mark the field an attached field that supplies a value to the reference field; used for vector and tagged fields

Type-Specific Import/Export Field Properties

	Type-Specific Import/Export Field Property	Explanation
Integer and Floating-Point Fields		
R	width = *nbytes* | max_width = *nbytes*	fixed or maximum text-format length
R	in_format = "*format"* out_format = "*format"* c_format = "*format"*	read and write using a C-style format string. For example: c_format="%3d"
R	big_endian | little_endian | native_endian	byte ordering; the default is native_endian
R	padchar = '*x*' | null	single-byte or unicode pad character for fixed-length fields (null = 0x00)

	Type-Specific Import/Export Field Property	Explanation
String Fields (string and ustring)		
R	width = *nbytes* | max_width = *nbytes*	fixed or maximum length
R	padchar = '*x*' | null	single-byte or unicode pad character for fixed-length fields (null = 0x00)
Decimal Fields		
R	width = *nbytes* | max_width = *nbytes*	fixed or maximum length
R	precision = *digits*	the number of digits (1 - 255)
R	scale = *digits*	number of decimal digits (0 precision)
R	nofix_zero | fix_zero	disallow (the default) or allow all-zero values
R	packed , check | nocheck , signed | unsigned	packed format; the defaults are check and signed
R	separate, leading | trailing	separate sign byte; the default is leading
R	zoned, leading | trailing	sign nibble; the default is trailing
R	round = ceil | floor | trunc_zero | round_inf	rounding type, where round_inf means the nearest value and trunc_zero is the default value
Date Fields		
R	date_format = "%yyyy-%mm-%dd"	date format string
R	julian	Julian day number (binary format)
R	days_since = "%yyyy-%mm-%dd"	days since date (binary format)
Time Fields		
R	big_endian | little_endian | native_endian	byte ordering; the default is native_endian
R	time_format = "%hh:%nn:%ss"	time format string
R	midnight_seconds	seconds since midnight (binary format)
Timestamp Fields		
R	big_endian | little_endian | native_endian	byte ordering; the default is native_endian
R	timestamp_format = "%yyyy-%mm-%dd %hh:%nn:%ss"	timestamp format string
Vector Fields		
	vector_prefix = *n*	*n*-byte vector length prefix where *n* = 1, 2, or 4

	Type-Specific Import/Export Field Property	Explanation
Nullable Fields		
	null_field = "string"	single-byte character string or unicode string representation of a null value. An example is: null_field = "NULL\0\0\0"
	null_length = *value* , actual_length = *nbytes*	field length indicating a null actual length to skip for a null (optional)

Generator Operator Field Properties Syntax

The **generator** operator chapter of the *Orchestrate 6.5 Operators Reference* contains more details about these properties.

Data Type	Generator Field Properties	Explanation

numeric
(also decimal, date, time, timestamp)
	`cycle = {init = `*init_val*`,`
	`  incr = `*incr_val*`, limit = `*limit_val*`}`
	`|`
	`random = {limit = `*limit_val*`,`
	`  seed = `*seed_val*`, signed}`
		generate a repeating pattern of values
		
		generate random values

date	`epoch = '`*date*`'`
	
	`invalids = `*percentage*
	
	`function = rundate`
		set the earliest date for the field; use `yyyy-mm-dd` format
		percentage of fields with invalid values
		set the field to the current date

decimal	`zeros = `*percentage* `invalids = `*percentage*	percentage of fields set to `0x00` percentage of fields set to `0xFF` (invalid)
raw	no options available	

string	`cycle = {value = '`*string_1*`',`
	`  value = '`*string_2*`', ... }`
	
	`alphabet = '`*alpha_numeric_string*`'`
		list of string values
		
		sets all charaters to successive characters in the string
		`s: string[3]`
		`  {alphabet = 'abc'};`
		produces `aaa bbb ccc aaa ...`

ustring	`cycle = {value = 'u`*string_1*`',`
	`  value = '`*ustring_2*`', ... }`
	
	`alphabet =`
	`'`*alpha_numeric_ustring*`'`
		list of ustring values
		
		sets all charaters to successive characters in the string
		`s: ustring[3]`
		`  {alphabet = 'abc'};`
		produces `aaa bbb ccc aaa ...`

Data Type	Generator Field Properties	Explanation
time	`scale = factor`	specify a multiplier to the increment value
	`invalids = percentage`	percentage of invalid field values
timestamp	`epoch = 'date'`	set the earliest date for the field; use `yyyy-mm-dd` format
	`scale = factor`	specify a multiplier to the increment value
	`invalids = percentage`	percentage of fields with invalid values
nullable	`nulls = percentage`	percentage of fields set to null
	`nullseed = number`	set the seed for the random number generator used to determine whether a field is null

Ascential Software Corporation
50 Washington Street
Westboro, MA 01581-1021
508 366-3888
508 389-8955 FAX

If you require assistance or have questions about Orchestrate, you can contact Ascential Customer Support by:

- Calling (866) INFONOW

- Writing support@ascential.com for any Ascential Software product or orch-support@ascential.com for Orchestrate-specific help.

- Logging onto the Ascential Support eService Web site at:

  www.ascential.com:8080/eservice/index.html

For current information about Ascential and its products log onto:

http://www.ascentialsoftware.com/