## Type Conversion

Special Python types for the BEAM Python API:

|  |  |
| --- | --- |
| CArray  beampy\_carray.h | A type used to represent content of native C-arrays in Python. Implements the *buffer protocol* ([http://docs.python.org/3.3/c-api/buffer.html](http://docs.python.org/3.3/c-api/buffer.html?highlight=buffer)). |
| JObject  beampy\_jobject.h | A type that wraps a Java object. Has a single private member “jobjectRef”. Base class for all other Java object representations in Python. Implements applicable parts of the Python *object protocol* (<http://docs.python.org/3.3/c-api/object.html>): PyObject\_ASCII, PyObject\_Str, PyObject\_Repr, PyObject\_IsInstance, PyObject\_IsSubclass, PyObject\_Hash, PyObject\_IsTrue, PyObject\_Not, PyObject\_RichCompare |
| JObjectArray(JObject)  generated with special sequence methods | A type that wraps a Java java.lang.Object[]. Supports the Python *sequence protocol* (<http://docs.python.org/3.3/c-api/sequence.html>). |
| String(JObject)  generated with special factory method | A type that wraps the immutable Java java.lang.String. |
| List(JObject)  generated with special sequence methods | A type that wraps a Java java.util.List object. Supports the Python sequence protocol (<http://docs.python.org/3.3/c-api/sequence.html>). |
| Map(JObject)  generated with special mapping methods | A type that wraps a Java java.util.Map object. Supports the Python *mapping protocol* (<http://docs.python.org/3.3/c-api/mapping.html>). |

### How Python objects are converted to JNI function arguments

Described here is how Python objects are converted to Java method parameters of type T. In the following RA means “Argument Conversion Rule”, GA means “Argument Conversion Group”.

General argument conversion rules:

|  |  |
| --- | --- |
| RA1: | If T is a **primitive type**, then Python arguments are converted to Java type T using the PyArg\_ParseTuple(args, format, &jarg1, &jarg2, …) Python function (see <http://docs.python.org/3.3/c-api/arg.html>). |
| RA2: | If T is an **Object** and the Python argument is None (type “NonType”), then the converted Java JNI value is NULL. |
| RA3: | If T is an **Object** and the type of the Python arg is “JObject” or one of its sub-types, then the conversion function is jarg = JObject\_ToJObjectRef(arg). In this case, T must be assignable from T, otherwise an error is raised. |
| RA4: | If T is a **primitive wrapper type** and neither rule RA2 nor RA3 applies, then PyArg\_ParseTuple() (as for RA1) is used to generate the primitive value which is then used to instantiate the Java argument. |
| RA5 | If T is a **primitive array type** and the Python is a T-compatible buffer (<http://docs.python.org/3.3/c-api/buffer.html?highlight=buffer>), then the conversion function is jarg = BPy\_NewJ<T>ArrayFromBuffer(arg). |
| RA6 | If T is a **primitive array** **type** and the Python object is a sequence, then the conversion function is jarg = BPy\_NewJ<T>ArrayFromSeq(arg). |

|  |  |  |  |
| --- | --- | --- | --- |
| GA | Java param type T | Python object types | Conversion function |
| GA1 | boolean | RA1 | PyArg\_ParseTuple(), format = “b” |
| char | “ | PyArg\_ParseTuple(), format = “C” |
| byte | “ | PyArg\_ParseTuple(), format = “b” |
| short | “ | PyArg\_ParseTuple(), format = “h” |
| int | “ | PyArg\_ParseTuple(), format = “i” |
| long | “ | PyArg\_ParseTuple(), format = “L” |
| float | “ | PyArg\_ParseTuple(), format = “f” |
| double | “ | PyArg\_ParseTuple(), format = “d” |
| GA 2 | java.lang.Boolean | RA2 or RA3 or RA4 | see RA2 or RA3 or RA4 |
| java.lang.Character | “ | “ |
| java.lang.Byte | “ | “ |
| java.lang.Short | “ | “ |
| java.lang.Integer | “ | “ |
| java.lang.Long | “ | “ |
| java.lang.Float | “ | “ |
| java.lang.Double | “ | “ |
| GA3 | java.lang.Object | RA2 or RA3 | see RA2 or RA3 |
| int | BPy\_NewJNumberFromInt(arg)(1) |
| float | BPy\_NewJNumberFromFloat(arg)(2) |
| str | BPy\_NewJStringFromStr(arg) |
| dict | BPy\_NewJMapFromDict(arg) |
| sequence | BPy\_NewJObjectArrayFromSeq(arg) (4) |
| GA4 | java.lang.String | RA2 or RA3 | see RA2 or RA3 |
| str | BPy\_NewJStringFromStr(arg) |
| GA5 | java.util.Map | RA2 or RA3 | see RA2 or RA3 |
| dict | BPy\_NewJMapFromDict(arg) (3) |
| GA6 | java.util.List | RA2 or RA3 | see RA2 or RA3 |
| sequence | BPy\_NewJListFromSeq(arg) (4) |
| GA7 | boolean[] | RA2 or RA3 or RA5 or RA6 | see RA2 or RA3 or RA5 or RA6 |
| char[] | “ | “ |
| byte[] | “ | “ |
| short[] | “ | “ |
| int[] | “ | “ |
| long[] | “ | “ |
| float[] | “ | “ |
| double[] | “ | “ |
| GA8 | java.lang.Object[] | RA2 or RA3 | see RA2 or RA3 |
| sequence | BPy\_NewJObjectArrayFromSeq(par) (5) |
| GA9 | java.lang.String[] | RA2 or RA3 | see RA2 or RA3 |
| sequence | BPy\_NewJStringArrayFromSeq(par) (6) |
| GA10 | T[] (any other type) | RA2 or RA3 | see RA2 or RA3 |
| sequence | BPy\_NewJDerivedObjectArrayFromSeq(par, &<T>\_Type) (7) |
| GA11 | T (any other type) | RA2 or RA3 | see RA2 or RA3 |

(1) Creates java.lang.Byte, Short, Integer, Long, BigInteger depending on the value of par.bit\_length().

(2) Creates java.lang.Float, Double depending on the value of par.bit\_length().

(3) The Java key and value objects of the of the new Map are derived using the same rules as if the Python keys values would be converted to Java arguments of type java.lang.Object. Default Map implementation used is java.util.HashMap.

(4) The Java element objects of the new array or List are derived using the same rules as if the Python sequence items would be converted to Java arguments of type java.lang.Object. Default List implementation used is java.util.ArrayList.

(5) The Java element objects of the new array are derived using the rules as if the Python sequence items would be converted to Java arguments of type java.lang.Object.

(6) The Java element objects of the new array are derived using the rules as if the Python sequence items would be converted to Java arguments of type java.lang.String.

### Summary of Argument Conversion Functions

#### Low Level

|  |  |  |
| --- | --- | --- |
| Function | Argument | Returns |
| JObject\_ToJObjectRef(arg) | **JObject** (RA3) | jobject |
| BPy\_NewJ<T>ArrayFromBuffer(arg) | T-compatible **buffer** (RA5) | jarray (e.g. int[]) |
| BPy\_NewJ<T>ArrayFromSeq(arg) | arg must be a **sequence** with T-compatible items (RA6) | jarray (e.g. int[]) |
| BPy\_NewJNumberFromInt(arg) | an **int** | jobject (e.g. java.lang.Short) |
| BPy\_NewJNumberFromFloat(arg) | a **float** | jobject |
| BPy\_NewJStringFromStr(arg) | a **str** | jstring |
| BPy\_NewJMapFromDict(arg) | a **mapping** (e.g. dict) | jobject (e.g. java.util.HashMap) |
| BPy\_NewJObjectArrayFromSeq(arg) | a **sequence** (e.g. list) | jobjectArray |
| BPy\_NewJListFromSeq(arg) | a **sequence** (e.g. list) | jobject (e.g. java.util.ArrayList) |

#### High Level

|  |  |  |  |
| --- | --- | --- | --- |
| GA | Function | Argument | Returns |
| GA3 | BPy\_ToJObject(arg) | any | jobject (java.lang.Object) |
| GA4 | BPy\_ToJString(arg) | None, JObject, str | jstring (java.lang.String) |
| GA5 | BPy\_ToJMap(arg) | None, JObject, mapping | jobject (java.util.Map) |
| GA6 | BPy\_ToJList(arg) | None, JObject, sequence | jobject (java.util.List) |
| GA7 | BPy\_ToJ<T>Array(arg) | None, JObject, sequence, buffer | jarray (T[]) |
| GA8 | BPy\_ToJObjectArray(arg) | None, JObject, sequence | jobjectArray |
| GA9 | BPy\_ToJStringArray(arg) | None, JObject, sequence | jobjectArray |
| GA10 | BPy\_ToDerivedObjectArray(arg, &<T>\_Type) | None, JObject, sequence | jobjectArray (T) |
| GA11 | BPy\_ToDerivedObject(arg, &<T>\_Type) | None, JObject | jobject (T) |

### How JNI function return values are converted to Python objects

Described here is how a Java (function return) value *res* of type T is converted to a corresponding Python object. In the following RR means “return-value conversion rule”, GR means “return-value conversion group”.

General rules:

|  |  |
| --- | --- |
| RR1: | If T is a primitive array, then … |

|  |  |  |  |
| --- | --- | --- | --- |
| GR | Java return type | Conversion function | Python type |
| GR1 | boolean | PyBool\_FromLong(res) | bool |
| char | PyUnicode\_FromFormat("%c", res) | str |
| byte | PyLong\_FromLong(res) | int |
| short | PyLong\_FromLong(res) | int |
| int | PyLong\_FromLong(res) | int |
| long | PyLong\_FromLongLong(res) | int |
| float | PyFloat\_FromDouble(res) | float |
| double | PyFloat\_FromDouble(res) | float |
| GR2 | java.lang.String | BPy\_NewStrFromJString(res) | str |
| GR3 | boolean[] | BPy\_NewCArrayFromJPrimArray(), format=”b” | RR1(1) |
| char[] | BPy\_NewCArrayFromJPrimArray(), format=”h” | “ |
| byte[] | BPy\_NewCArrayFromJPrimArray(), format=”b” | “ |
| short[] | BPy\_NewCArrayFromJPrimArray(), format=”h” | “ |
| int[] | BPy\_NewCArrayFromJPrimArray(), format=”i” | “ |
| long[] | BPy\_NewCArrayFromJPrimArray(), format=”l” | “ |
| float[] | BPy\_NewCArrayFromJPrimArray(), format=”f” | “ |
| double[] | BPy\_NewCArrayFromJPrimArray(), format=”d” | “ |
| GR4 | java.lang.String[] | BPy\_NewListFromJStringArray(res) | list of str |
| GR5 | <T>[] | JObjectArray\_New(res) | JObjectArray(2) |
| GR6 | <T> | JObject\_FromType(&<T>\_Type, res) | JObject |

(1) The return value may be identical to one of the parameters (return parameter). The format string I the one used by CArray\_FromMemory()

(2) The JObjectArray Python type implements the sequence protocol and parts of the list protocol.