Middleware

Petr Tůma

Middleware

Petr Tůma

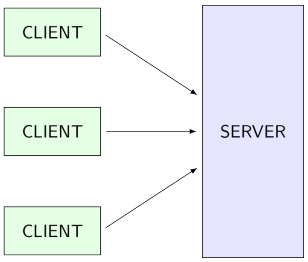
This is version c1ed93857a3938ab7a4698a9e338d7f17033b882 (modified) generated on 2025-05-12 08:50:35. For the latest version, check http://d3s.mff.cuni.cz/teaching/nswi080.

Table of Contents

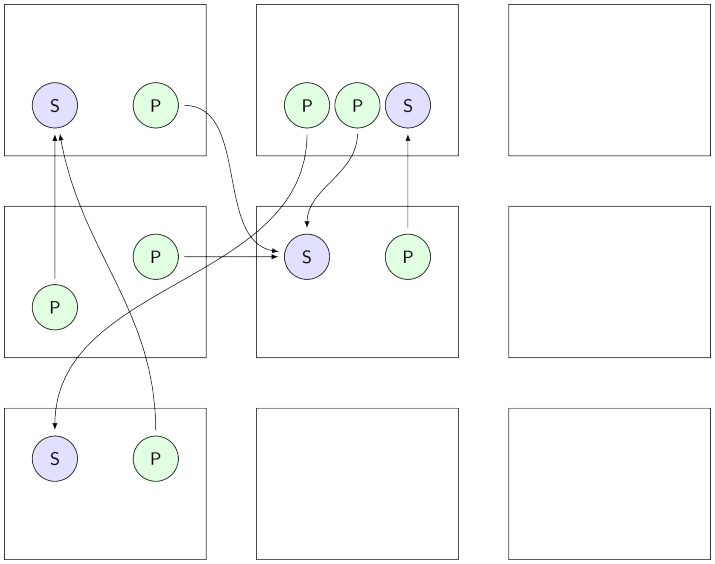
Chapter 1. Concepts

1.1. Architectures

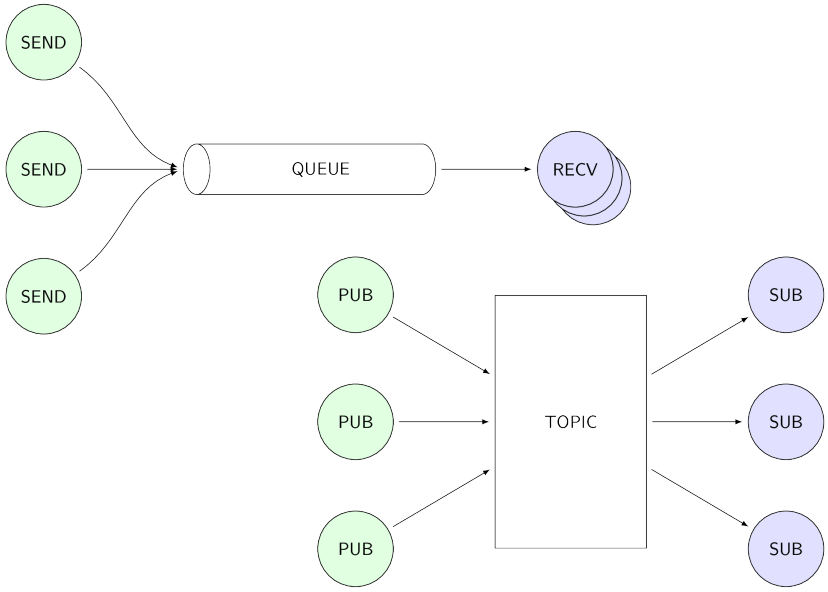
1.1.1. Client-Server



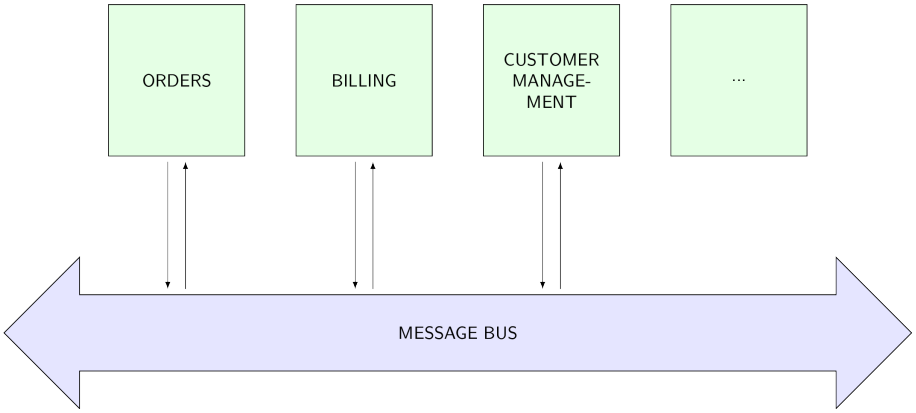
1.1.2. Distributed Objects



1.1.3. Messaging



1.1.4. Message Bus



1.2. Serialization

1.2.1. Textual

1.2.1.1. XML Object Serialization Example

Data Instance.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<anExampleDataClass>
    <anIntField>123</anIntField>
    <aFloatField>12.34</aFloatField>
    <aDoubleField>1.234E57</aDoubleField>
    <aBoxedIntField>987</aBoxedIntField>

    <aRequiredStringField>a string</aRequiredStringField>
    <anArrayWithoutAWrapper>1</anArrayWithoutAWrapper>
    <anArrayWithoutAWrapper>2</anArrayWithoutAWrapper>
    <anArrayWithoutAWrapper>3</anArrayWithoutAWrapper>

    <anArrayWithAWrapper>
        <anArrayElement>12</anArrayElement>
        <anArrayElement>34</anArrayElement>
        <anArrayElement>56</anArrayElement>
    </anArrayWithAWrapper>

    <aListElement>
        <anIntField>0</anIntField>
        <aFloatField>0.0</aFloatField>
        <aDoubleField>0.0</aDoubleField>
    </aListElement>

    <aSetElement>
        <anIntField>0</anIntField>
        <aFloatField>0.0</aFloatField>
        <aDoubleField>0.0</aDoubleField>
    </aSetElement>

    <aMapElement>
        <entry>
```

```
            <key>456</key>
            <value>
                <anIntField>0</anIntField>
                <aFloatField>0.0</aFloatField>
                <aDoubleField>0.0</aDoubleField>
            </value>
        </entry>
        <entry>
            <key>123</key>
            <value>
                <anIntField>0</anIntField>
                <aFloatField>0.0</aFloatField>
                <aDoubleField>0.0</aDoubleField>
            </value>
        </entry>
    </aMapElement>
</anExampleDataClass>
```

Possible Schema.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">

    <xs:element name="anExampleDataClass" type="anExampleDataClass"/>

    <xs:complexType name="anExampleDataClass">
        <xs:annotation>
            <xs:documentation>
                An example class.
                Contains various field types to illustrate the mapping.
            </xs:documentation>
        </xs:annotation>

        <xs:all>
            <xs:element name="anIntField" type="xs:int"/>
            <xs:element name="aFloatField" type="xs:float"/>
            <xs:element name="aDoubleField" type="xs:double"/>

            <xs:element minOccurs="0" name="aBoxedIntField" type="xs:int"/>
            <xs:element name="aRequiredStringField" type="xs:string"/>
            <xs:element minOccurs="0" name="anOptionalStringField" type="xs:str
            <xs:element default="default" minOccurs="0" name="aStringFieldWithD

            <xs:element maxOccurs="unbounded" minOccurs="0" name="anArrayWithou
            <xs:element minOccurs="0" name="anArrayWithAWrapper">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element maxOccurs="unbounded" minOccurs="0" name="a
                    </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element maxOccurs="unbounded" minOccurs="0" name="aListElement"
            <xs:element maxOccurs="unbounded" minOccurs="0" name="aSetElement" 

            <xs:element name="aMapElement">
                <xs:complexType>
```

```
                    <xs:sequence>
                        <xs:element maxOccurs="unbounded" minOccurs="0" name="e
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element minOccurs="0" name="key" type="
                                    <xs:element minOccurs="0" name="value" type
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:all>
    </xs:complexType>
</xs:schema>
```

1.2.1.2. JSON Object Serialization Example

Data Instance.

```
{
    "an_int_field" : 123,
    "a_float_field" : 12.34,
    "a_string_field" : "a string",
    "an_array" : [1, 2, 3]
}
```

1.2.1.3. YAML Object Serialization Example

Data Instance.

```
an_int_field: 123
a_float_field: 12.34
a_string_field: a string
an_array:
- 1
- 2
- 3
a_mapping_field:
    &some_name a_nested_field: a string
    a_reference: *some_name
...
```

1.2.2. Binary

1.2.2.1. Concise Binary Object Representation (CBOR)

1.2.2.1.1. CBOR Serialization

Integer	Types 0 (positive) and 1 (negative). Argument values 0-23 encode integers directly. Argument values 24-27 indicate integers in the next 1, 2, 4, 8 bytes big endian.
String	Types 2 (binary) and 3 (UTF-8). Argument value encoded as integer gives string length.
Array	Type 4. Argument value encoded as integer gives array length. Individual elements encoded with own headers.

Map	Type 5. Argument value encoded as integer gives map length. Individual key value pairs encoded with own headers.
Tag	Type 6. Tag value encoded as integer gives special meaning to next item. This includes date and time, big number binary strings, URL, special arrays and more.
Simple or Float	Type 7. Argument value indicate simple item or one of IEEE 754 formats. Simple items include booleans, null, NA and indefinite sequence stop mark.

1.2.2.1.2. CBOR Serialization Examples

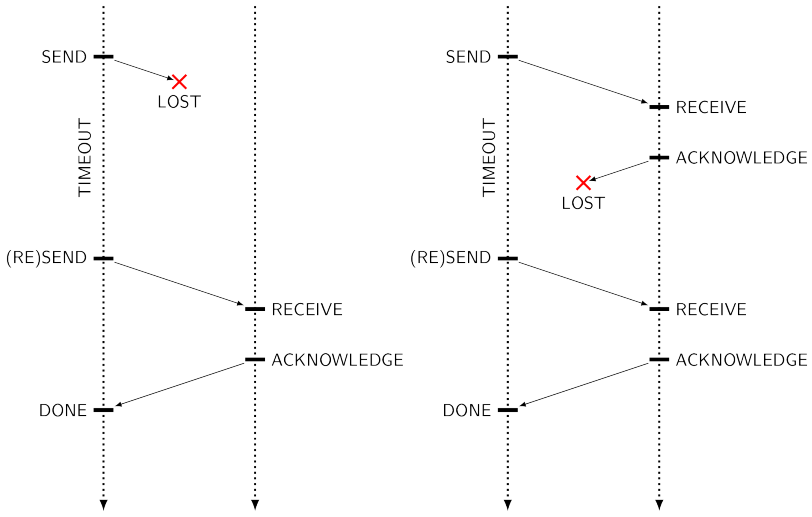
Integer Data Items.

```
00h ~ 000-00000b ~ positive integer type (0) value 0
01h ~ 000-00001b ~ positive integer type (0) value 1
17h ~ 000-10111b ~ positive integer type (0) value 23

18h ~ 000-11000b ~ positive integer type (0) value in next byte (24)
18h              ~ value 24 (18h)
18h ~ 000-11000b ~ positive integer type (0) value in next byte (24)
19h              ~ value 25 (19h)

19h ~ 000-11001b ~ positive integer type (0) value in next two bytes (25)
01h 00h          ~ value 256 (big endian)

1Ah ~ 000-11010b ~ positive integer type (0) value in next four bytes (26)
00h 01h 00h 00h  ~ value 65536 (big endian)

20h ~ 001-00000b ~ negative integer type (1) value -1
21h ~ 001-00001b ~ negative integer type (1) value -2

38h ~ 001-11000b ~ negative integer type (1) value in next byte (24)
FFh              ~ value -256
```

1.3. Protocols

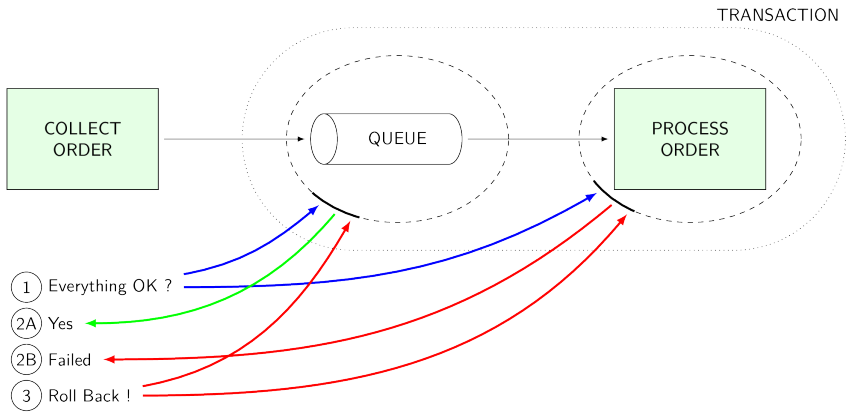
1.3.1. Reliability

1.3.1.1. Message Acknowledgment

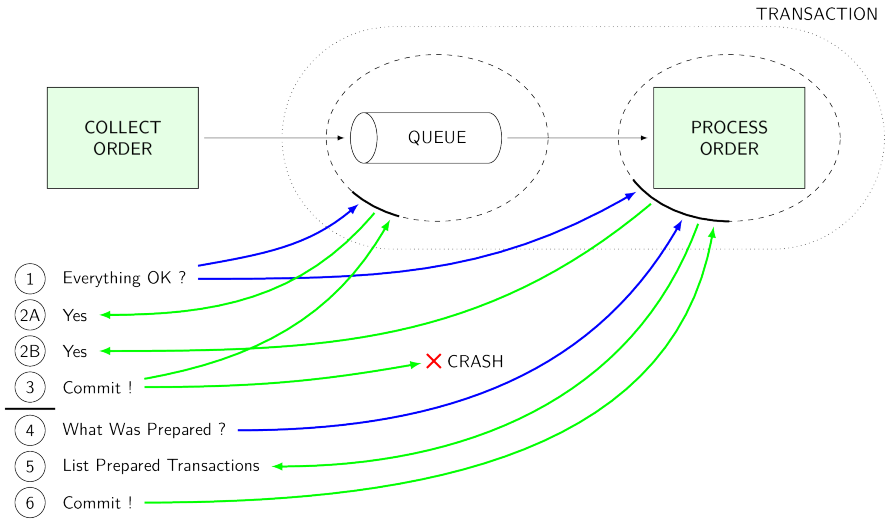


1.3.2. Atomicity

1.3.2.1. Transactional Messaging Rollback

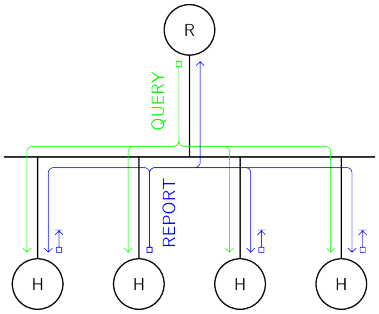


1.3.2.2. Transactional Messaging Commit



1.3.3. Multicast Membership

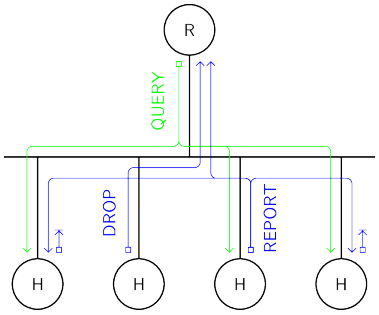
1.3.3.1. Group Membership Query



Protocol.

QUERY	General Multicast Listener Query periodically multicast by router
REPORT	Multicast Listener Report multicast after random delay

1.3.3.2. Group Membership Done

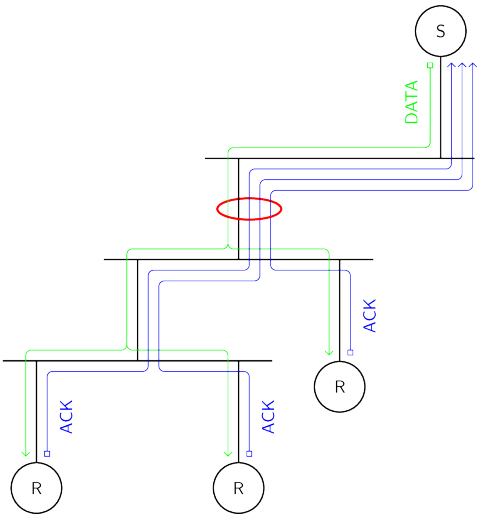


Protocol.

DROP	Multicast Listener Drop sent from host to routers
QUERY	Specific Multicast Listener Query multicast by router
REPORT	Multicast Listener Report multicast after random delay

1.3.4. Multicast Reliability

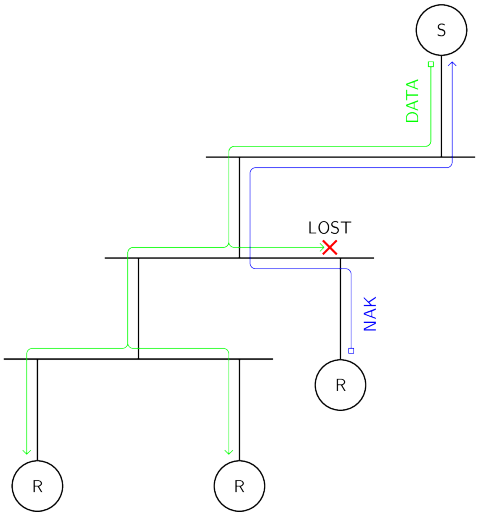
1.3.4.1. Multicast Sender Initiated Error Recovery



Features.

- Can suffer from ACK implosion
- Sender must know all receivers
- Sender knows when data can be dropped

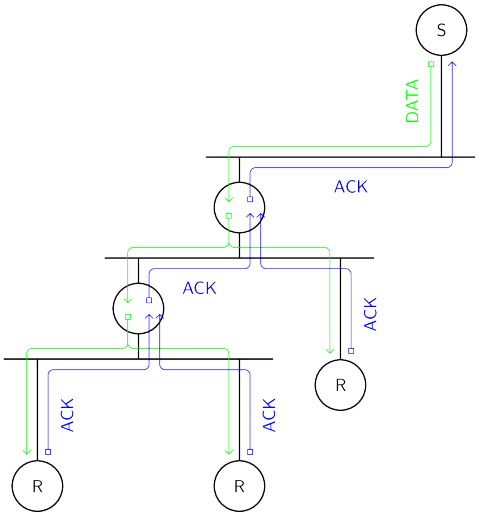
1.3.4.2. Multicast Receiver Initiated Error Recovery



Features.

- Can suffer from NAK implosion
- Sender must transmit keepalive messages.
- Sender does not know when data can be dropped

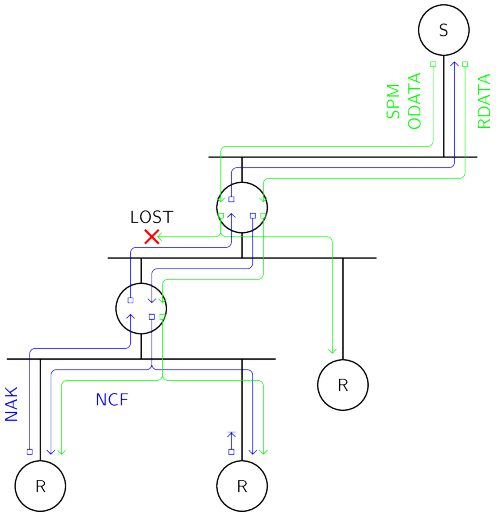
1.3.4.3. Aggregated Multicast Error Recovery



Features.

- Multiple variants with different acknowledgments possible
- Requires cooperation from network elements
- Can be substituted with overlay network

1.3.4.4. Pragmatic General Multicast



Protocol.

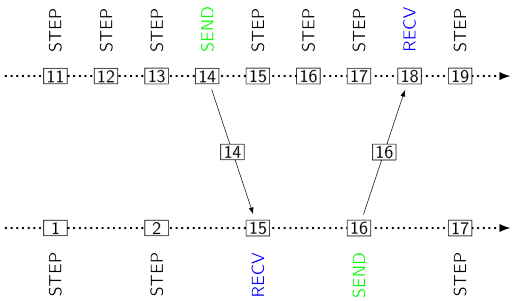
SPM	Source Path Messages establish path information and perform keepalive function
ODATA	Original data packets multicast to all receivers
NAK	Negative Acknowledgment unicast to nearest parent along path
NCF	Negative Acknowledgment Confirmation multicast to children along path
RDATA	Repair data packets multicast to selected receivers

1.3.5. Multicast Ordering

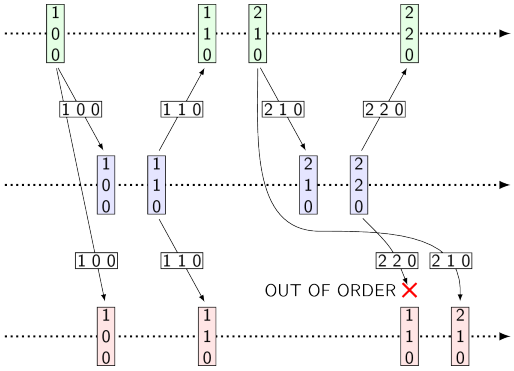
1.3.5.1. Message Ordering

Source Ordering	Each node defines local order of SEND operations
	Message delivery observes union of the local orderings
Causal Ordering	Each node defines local order of SEND operations
	Each node defines local order of RECV-SEND operation pairs
	Message delivery semantics defines global order of SEND-RECV operation pairs
	Message delivery observes transitive closure of the orderings
Total Ordering	All nodes observe the same order of SEND and RECV operations

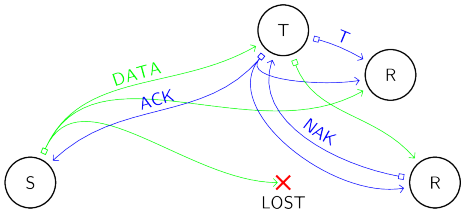
1.3.5.2. Lamport Clock



1.3.5.3. Vector Clock



1.3.5.4. Token Ring Based Multicast



Chapter 2. Systems

2.1. CORBA

2.1.1. Interface Definition Language

2.1.1.1. Basic Types

2.1.1.1.1. Integer Types

short	16 bit signed integer
long	32 bit signed integer
long long	64 bit signed integer
unsigned short	16 bit unsigned integer
unsigned long	32 bit unsigned integer
unsigned long long	64 bit unsigned integer

Values.

```
18, 022, 0x12, 0X12
```

Constants.

```
const short aShortConstant = 6 * 7;
```

2.1.1.1.2. Floating Point Types

float	24 bit signed fraction, 8 bit signed exponent
double	53 bit signed fraction, 11 bit signed exponent
long double	113 bit signed fraction, 15 bit signed exponent

Values.

```
3.14, 12.34e5, 1.2E-4
```

Constants.

```
const float aFloatConstant = 3.141593;
```

2.1.1.1.3. Character Types

char	character in single-byte character set
wchar	character in multiple-byte character set

Values.

```
'a', '\n', '\000', '\x12'
```

Constants.

```
const char aTab = '\t';
const wchar aWideTab = L'\t';
```

2.1.1.1.4. Logical Types

boolean	logical value

Values.

```
TRUE, FALSE
```

Constants.

```
const boolean aTrueValue = TRUE;
const boolean aFalseValue = FALSE;
```

2.1.1.1.5. Special Types

octet	8 bits of raw data
any	container of another arbitrary type

2.1.1.2. Constructed Data Types

2.1.1.2.1. Structures

Declaration.

```
struct aPerson
{
  string firstName;
  string lastName;
  short  age;
};
```

2.1.1.2.2. Exceptions

Declaration.

```
exception anException
{
  string reason;
  string severity;
};
```

Standard System Exception.

```
exception COMM_FAILURE
{
  unsigned long minor;
  completion_status completed;
};
```

2.1.1.2.3. Unions

Declaration.

```
union aSillyUnion switch (short)
{
```

```
  case 1  : long aLongValue;
  case 2  : float aFloatValue;
  default : string aStringValue;
};
```

2.1.1.2.4. Enums

Declaration.

```
enum aBaseColor { red, green, blue }
```

2.1.1.2.5. Arrays

Declaration.

```
typedef long aLongArray [10];
```

2.1.1.2.6. Sequences

Declaration.

```
typedef sequence<long,10> aBoundedVector;
typedef sequence<long> anUnboundedVector;
```

2.1.1.2.7. Strings

Declaration.

```
typedef string<10> aBoundedString;
typedef string anUnboundedString;
```

Constants.

```
const string aHello = "Hello\n";
const wstring aWideHello = L"Hello\n";
```

2.1.1.2.8. Fixed Point Types

Declaration.

```
typedef fixed<10,2> aPrice;
```

Constants.

```
const fixed aPrice = 12.34D;
```

2.1.1.3. Constructed Object Types

2.1.1.3.1. Interface Types

Declaration.

```
abstract interface aParentInterface
{
```

```
  attribute string aStringAttribute;
  short aMethod (in long aLongArgument, inout float aFloatArgument);
}

interface aChildInterface : aParentInterface
{
  readonly attribute short aShortAttribute;
  oneway void aOnewayMethod (in long anArgument);
  void aTwowayMethod () raises anException;
}
```

Keywords.

local	interface not invoked remotely
abstract	runtime determines passing semantics
oneway	best effort delivery
readonly	attribute without setter

2.1.1.3.2. Value Types

Declaration.

```
valuetype aChildValue : truncatable aParentValue, supports anInterface
{
  private short aShortMember;
  public aParentValue aValueMember;
  factory aFactory (in string anArgument);
  short aLocalMethod (in long aLongArgument, in float aFloatArgument);
}
```

Keywords.

custom	custom marshalling
abstract	base type not instantiated
truncatable	state compatible with parent
public	value used by clients
private	value used by implementation
factory	portable initializer

2.1.2. Language Mapping

2.1.2.1. Integer And Floating Point Types

2.1.2.1.1. Holder Class Example

```
public final class IntHolder
implements org.omg.CORBA.portable.Streamable {
    public int value;

    public IntHolder () { }
    public IntHolder (int o) { value = o; }

    public TypeCode _type () {
        return ORB.init ().get_primitive_tc (TCKind.tk_long);
    }
```

```
    public void _read (org.omg.CORBA.portable.InputStream in) {
        value = in.read_long ();
    }

    public void _write (org.omg.CORBA.portable.OutputStream out) {
        out.write_long (value);
    }
}
```

2.1.2.2. Character And String Types

2.1.2.2.1. Var Class Example

```
class String_var {
private:

    char *data;

public:

    inline String_var ()        { data = 0; }
    inline String_var (char *p) { data = p; }

    inline String_var (const char *p) {
        if (p) data = CORBA::string_dup (p);
        else   data = 0;
    }

    inline ~String_var () {
        CORBA::string_free (data);
    }

    inline String_var &operator = (char *p) {
        CORBA::string_free (data);
        data = p;
        return (*this);
    }

    inline operator char * () { return (data); }

    inline char &operator [] (CORBA::ULong index) {
        return (data [index]);
    }

    ...
}
```

2.1.2.2.2. Var Class Usage

```
void FunctionWithoutLeaks (void) {
    // All strings must be allocated using specific functions
    String_var vSmartPointer = string_dup ("A string ...");

    // Except assignment from const string which copies
    const char *pConstPointer = "A const string ...";
```

```
    vSmartPointer = pConstPointer;

    // Assignment releases rather than overwrites
    vSmartPointer = string_dup ("Another string ...");

    // Going out of scope releases too
    throw (0);
}
```

2.1.2.3. Any Type

2.1.2.3.1. Any Class Example

```
class Any {
public:

    // Types passed by value are easy
    void operator <<= (Any &, Short);
    Boolean operator >>= (const Any &, Short &);
    ...

    // Types passed by reference introduce ownership issues
    void operator <<= (Any &, const Any &);
    void operator <<= (Any &, Any *);
    ...

    // Types where overloading fails introduce resolution issues
    struct from_boolean { from_boolean (Boolean b) : val (b) { } Boolean val; }
    struct from_octet { from_octet (Octet o) : val (o) { } Octet val; };
    struct from_char { from_char (Char c) : val (c) { } Char val; };
    ...

    void operator <<= (from_boolean);
    void operator <<= (from_octet);
    void operator <<= (from_char);
    ...

    struct to_boolean { to_boolean (Boolean &b) : ref (b) { } Boolean &ref; };
    ...

    Boolean operator >>= (to_boolean) const;
    ...

private:

    // Private operators can detect resolution issues
    unsigned char void operator <<= (unsigned char);
    Boolean operator >>= (unsigned char &) const;
}
```

2.1.2.3.2. Any Class Insertion

```
Any oContainer;

// Small types can be stored easily
Long iLongValue = 1234;
```

```
Float fFloatValue = 12.34;
oContainer <<= iLongValue;
oContainer <<= fFloatValue;

// Constant references have copying semantics
const char *pConstString = "A string ...";
oContainer <<= pConstString;

// Non constant references have adoption semantics
String_var vString = string_dup ("A string ...");
oContainer <<= Any::from_string (vString, 0, FALSE);
oContainer <<= Any::from_string (vString._retn (), 0, TRUE);

// Some types need to be resolved explicitly
Char cChar = 'X';
Octet bOctet = 0x55;
oContainer <<= Any::from_char (cChar);
oContainer <<= Any::from_octet (bOctet);
```

2.1.2.3.3. Any Class Extraction

```
Any oContainer;

// Small types can be retrieved easily
Long iLongValue;
Float fFloatValue;
if (oContainer >>= iLongValue) ...;
if (oContainer >>= fFloatValue) ...;

// References remain owned by container
const char *pConstString;
if (oContainer >>= Any::to_string (pConstString, 0)) ...;

// Some types need to be resolved explicitly
Char cChar;
Octet bOctet;
if (oContainer >>= Any::to_char (cChar)) ...;
if (oContainer >>= Any::to_octet (bOctet)) ...;
```

2.1.2.4. Structures And Exceptions

2.1.2.4.1. Exception Class Example

```
class Exception {
public:

    // Method for throwing most derived type
    virtual void _raise () const = 0;
    ...
}
```

2.1.2.5. Unions

2.1.2.5.1. Union Class Example

```
class AUnion {
public:

    ...

    void _d (Short);    // Set discriminator
    Short _d() const;   // Get discriminator

    void ShortItem (Short);     // Store ShortItem and set discriminator
    Short ShortItem () const;   // Read ShortItem if stored

    void LongItem (Long);       // Store LongItem and set discriminator
    Long LongItem () const;     // Read LongItem if stored

    ...
}
```

2.1.2.5.2. Union Class Usage

```
AUnion oUnion;
Short iShortValue = 1234;
Long iLongValue = 5678;

// Storing sets discriminator
oUnion.ShortItem (iShortValue);
oUnion.LongItem (iLongValue);

// Retrieving must check discriminator
if (oUnion._d () == 1) iShortValue = oUnion.ShortItem ();
if (oUnion._d () == 2) iLongValue = oUnion.LongItem ();
```

2.1.2.6. Enum Types

2.1.2.6.1. Enum Class Example

```
public class AnEnum {
    public static final int _red = 0;
    public static final AnEnum red = new AnEnum (_red);

    public static final int _green = 1;
    public static final AnEnum green = new AnEnum (_green);

    ...

    public int value () {...};
    public static AnEnum from_int (int value) {...};
}
```

2.1.2.6.2. Enum Class Usage

```
AnEnum oEnum;

// Assignments are type safe
oEnum = AnEnum.red;
oEnum = AnEnum.green;
```

```
// Switch statements use ordinal values
switch (oEnum.value ()) {
    case AnEnum._red: ...;
    case AnEmum._green: ...;
}
```

2.1.2.7. Sequences

2.1.2.7.1. Sequence Class Example

```
class ASequence {
public:

    ASequence ();
    ASequence (ULong max);
    ASequence (ULong max, ULong length, Short *data, Boolean release = FALSE);

    ...

    ULong maximum () const;
    Boolean release () const;

    void length (ULong);
    ULong length () const;

    T &operator [] (ULong index);
    const T &operator [] (ULong index) const;

    ...
}
```

2.1.2.8. Fixed Point Types

2.1.2.8.1. Fixed Class Example

```
class Fixed {
public:

    // Constructors

    Fixed (Long val);
    Fixed (ULong val);
    Fixed (LongLong val);
    Fixed (ULongLong val);
    ...
    Fixed (const char *);

    // Conversions

    operator LongLong () const;
    operator LongDouble () const;
    Fixed round (UShort scale) const;
    Fixed truncate (UShort scale) const;

    // Operators
```

```
    Fixed &operator = (const Fixed &val);
    Fixed &operator += (const Fixed &val);
    Fixed &operator -= (const Fixed &val);
    ...
}

Fixed operator + (const Fixed &val1, const Fixed &val2);
Fixed operator - (const Fixed &val1, const Fixed &val2);
...
```

2.1.2.9. Proxies

2.1.2.9.1. Proxy Interface Class Example

```
class AnInterface;
typedef AnInterface *AnInterface_ptr;
class AnInterface_var;


class AnInterface : public virtual Object {
public:

    typedef AnInterface_ptr _ptr_type;
    typedef AnInterface_var _var_type;

    static AnInterface_ptr _duplicate (AnInterface_ptr obj);
    static AnInterface_ptr _narrow (Object_ptr obj);
    static AnInterface_ptr _nil ();

    virtual ... AnOperation (...) = 0;

protected:

    AnInterface ();
    virtual ~AnInterface ();

    ...
}
```

2.1.2.9.2. Proxy Var Class Example

```
class AnInterface_var : public _var {
protected:

    AnInterface_ptr ptr;

public:

    AnInterface_var () { ptr = AnInterface::_nil (); }
    AnInterface_var (AnInterface_ptr p) { ptr = p; }

    ...

    ~AnInterface_var () {
        release (ptr);
```

```
        }

        AnInterface_var &operator = (AnInterface_ptr p) {
            release (ptr);
            ptr = p;
            return (*this);
        }

        AnInterface_var &operator = (const AnInterface_var &var) {
            if (this != &var) {
                release (ptr);
                ptr = AnInterface::_duplicate (AnInterface_ptr (var));
            }
            return (*this);
        }

        operator AnInterface_ptr & () { return (ptr); }
        AnInterface _ptr operator -> () const { return (ptr); }

        ...
    }
```

2.1.2.9.3. Proxy Class Example

```
    public interface AnInterfaceOperations {
        ... AnOperation (...) throws ...;
    }

    public interface AnInterface extends AnInterfaceOperations ... { }

    abstract public class AnInterfaceHelper {
        public static void insert (Any a, AnInterface t) {...}
        public static AnInterface extract (Any a) {...}
        public static AnInterface read (InputStream is) {...}
        public static void write (OutputStream os, AnInterface val) {...}
        ...

        public static AnInterface narrow (org.omg.CORBA.Object obj) {...}
        public static AnInterface narrow (java.lang.Object obj) {...}
    }

    final public class AnInterfaceHolder implements Streamable {
        public AnInterface value;
        public AnInterfaceHolder () { }
        public AnInterfaceHolder (AnInterface initial) {...}

        ...
    }
```

2.1.2.10. Servants

2.1.2.10.1. Servant Base Class

```
    class ServantBase {
    public:
```

```
    virtual ~ServantBase ();

    virtual InterfaceDef_ptr _get_interface () throw (SystemException);
    virtual Boolean _is_a (const char *logical_type_id) throw (SystemException)
    virtual Boolean _non_existent () throw (SystemException);

    virtual void _add_ref ();
    virtual void _remove_ref ();

    ...
}
```

2.1.2.10.2. Servant Class Example

```
class POA_AnInterface : public virtual ServantBase {
public:

    virtual ... AnOperation (...) = 0;

    ...
}

template <class T> class POA_AnInterface_tie : public POA_AnInterface {
public:

    POA_AnInterface_tie (T &t) : _ptr (t) { }

    ...

    ... AnOperation (...) { return (_ptr->AnOperation (...); }
}
```

2.1.2.10.3. Servant Base Class

```
class Servant {
public:

    virtual IDL::traits<CORBA::InterfaceDef>::ref_type _get_interface ();
    virtual bool _is_a (const std::string &logical_type_id);
    virtual bool _non_existent ();

    ...

protected:

    virtual ~Servant ();
}
```

2.1.2.10.4. Servant Class Example

```
class _AnInterface_Servant_Base : public virtual Servant {
public:

    virtual ... AnOperation (...) = 0;

    ...
```

```
}

class AnInterface_Servant : public virtual CORBA::servant_traits<AnInterface>::
public:

    virtual ... AnOperation (...) override;
}
```

2.1.2.10.5. Servant Base Class

```
abstract public class Servant {
    final public Delegate _get_delegate () { ... }
    final public void _set_delegate (Delegate delegate) { ... }
    ...
}
```

2.1.2.10.6. Servant Class Example

```
abstract public class AnInterfacePOA implements AnInterfaceOperations {
    public AnInterface _this () { ... }
    ...
}

public class AnInterfacePOATie extends AnInterfacePOA {
    private AnInterfaceOperations _delegate;

    public AnInterfacePOATie (AnInterfaceOperations delegate)
    { _delegate = delegate; }

    public AnInterfaceOperations _delegate ()
    { return (_delegate); }

    public void _delegate (AnInterfaceOperations delegate)
    { _delegate = delegate; }

    public ... AnOperation (...) { return (_delegate.AnOperation (...); }
}
```

2.1.2.11. Value Types

2.1.2.11.1. Value Mapping Example

```
class AValue : public virtual ValueBase {
public:

    virtual void ShortItem (Short) = 0;
    virtual Short ShortItem () const = 0;

    virtual void LongItem (Long) = 0;
    virtual Long LongItem () const = 0;

    ...

    virtual ... AnOperation (...) = 0;
}
```

```
class OBV_AValue : public virtual AValue {
public:

    virtual void ShortItem (Short) { ... };
    virtual Short ShortItem () const { ... };

    virtual void LongItem (Long) { ... };
    virtual Long LongItem () const { ... };

    ...

    virtual ... AnOperation (...) = 0;
}

class ValueFactoryBase {
private:

    virtual ValueBase *create_for_unmarshal () = 0;

    ...
}

class AValue_init : public ValueFactoryBase {
public:

    virtual AValue *AConstructor (...) = 0;

    ...
}
```

2.1.3. Object Adapter

2.1.3.1. Object Adapter Configuration

```
local interface POA {
  POA create_POA (in string adapter_name,
                  in POAManager manager,
                  in CORBA::PolicyList policies);

    ThreadPolicy create_thread_policy (in ThreadPolicyValue value);
    LifespanPolicy create_lifespan_policy (in LifespanPolicyValue value);
    ServantRetentionPolicy create_servant_retention_policy (in ServantRetentionP
    RequestProcessingPolicy create_request_processing_policy (in RequestProcessi

    ...
};

local interface POAManager {
    enum State { HOLDING, ACTIVE, DISCARDING, INACTIVE };
    State get_state ();

    void activate () raises (AdapterInactive);
    void hold_requests (in boolean wait_for_completion) raises (AdapterInactive
    void discard_requests (in boolean wait_for_completion) raises (AdapterInacti
```

```
    void deactivate (in boolean etherealize_objects,
                     in boolean wait_for_completion);
};
```

2.1.3.2. Thread Policy

Thread Policy Values.

SINGLE_THREAD_MODEL	calls to servants and managers are serialized
MAIN_THREAD_MODEL	calls to servants are using single main thread
ORB_CTRL_MODEL	calls use arbitrary threading model

2.1.3.3. Object Identity Policies

ID Uniqueness Policy Values.

UNIQUE_ID	servants have exactly one object ID
MULTIPLE_ID	servants have at least one object ID

ID Assignment Policy Values.

USER_ID	object ID is assigned by application
SYSTEM_ID	object ID is assigned by object adapter

Implicit Activation Policy Values.

IMPLICIT_ACTIVATION	assign object ID on demand
NO_IMPLICIT_ACTIVATION	do not assign object ID on demand

2.1.3.4. Object Activation

```
ObjectId activate_object (in Servant servant) raises (ServantAlreadyActive, Wro

void activate_object_with_id (in ObjectId oid, in Servant servant)
raises (ObjectAlreadyActive, ServantAlreadyActive, WrongPolicy);

void deactivate_object (in ObjectId oid) raises (ObjectNotActive, WrongPolicy);

Object create_reference (in CORBA::RepositoryId ifc) raises (WrongPolicy);
Object create_reference_with_id (in ObjectId oid, in CORBA::RepositoryId ifc);

Object servant_to_reference (in Servant servant) raises (ServantNotActive, Wrong
Servant reference_to_servant (in Object reference) raises (ObjectNotActive, Wro
```

2.1.3.5. Current Object Interface

```
local interface Current {
    POA get_POA () raises (NoContext);
    ObjectId get_object_id () raises (NoContext);
    Object get_reference () raises (NoContext);
    Servant get_servant () raises (NoContext);
};
```

2.1.3.6. Servant Lookup Policies

Servant Retention Policy Values.

RETAIN	keep track of active servants
NON_RETAIN	do not keep track of active servants

Request Processing Policy Values.

USE_ACTIVE_OBJECT_MAP_ONLY	only deliver to tracked servants
USE_DEFAULT_SERVANT	alternatively deliver to default servant
USE_SERVANT_MANAGER	alternatively activate servants on demand

2.1.3.7. Servant Activator Interface

```
local interface ServantActivator : ServantManager {

    Servant incarnate (in ObjectId oid,
                       in POA adapter)
    raises (ForwardRequest);

    void etherealize (in ObjectId oid,
                      in POA adapter,
                      in Servant servant,
                      in boolean cleanup_in_progress,
                      in boolean remaining_activations};
};
```

2.1.3.8. Servant Locator Interface

```
local interface ServantLocator : ServantManager {

    native Cookie;

    Servant preinvoke (in ObjectId oid,
                       in POA adapter,
                       in CORBA::Identifier operation,
                       out Cookie cookie)
    raises (ForwardRequest);

    void postinvoke (in ObjectId oid,
                     in POA adapter,
                     in CORBA::Identifier operation,
                     in Cookie cookie,
                     in Servant servant);
};
```

2.1.3.9. Lifespan Policy

Lifespan Policy Values.

TRANSIENT	object references have lifetime of object adapter
PERSISTENT	object references have potentially unlimited lifetime

2.1.3.10. Request Forward Exception

```
exception ForwardRequest {
    Object forward_reference;
```

```
};
```

2.1.4. Messaging

2.1.4.1. Synchronization Scope Policy

SYNC_NONE
SYNC_WITH_TRANSPORT
SYNC_WITH_SERVER
SYNC_WITH_TARGET

2.1.4.2. Routing Policy

ROUTE_NONE
ROUTE_FORWARD
ROUTE_STORE_AND_FORWARD

2.1.4.3. Asynchronous Messaging Mapping Example

Interface.

```
interface StockManager {
    attribute string stock_exchange_name;
    boolean add_stock (in string symbol, in double quote);
    void remove_stock (in string symbol, out double quote) raises (InvalidStock
};
```

Callback Mapping.

```
void sendc_get_stock_exchange_name (
    in AMI_StockManagerHandler ami_handler);
void sendc_set_stock_exchange_name (
    in AMI_StockManagerHandler ami_handler,
    in string attr_stock_exchange_name);

void sendc_add_stock (
    in AMI_StockManagerHandler ami_handler,
    in string symbol,
    in double quote);

void sendc_remove_stock (
    in AMI_StockManagerHandler ami_handler,
    in string symbol);


interface AMI_StockManagerHandler : Messaging::ReplyHandler {
    void get_stock_exchange_name (
        in string ami_return_val);
    void get_stock_exchange_name_excep (
        in Messaging::ExceptionHolder excep_holder);

    void set_stock_exchange_name ();
    void set_stock_exchange_name_excep (
        in Messaging::ExceptionHolder excep_holder);
```

```
    void add_stock (in boolean ami_return_val);
    void add_stock_excep (
        in Messaging::ExceptionHolder excep_holder);

    void remove_stock (in double quote);
    void remove_stock_excep (
        in Messaging::ExceptionHolder excep_holder);
};
```

Poller Mapping.

```
AMI_StockManagerPoller sendp_get_stock_exchange_name ();
AMI_StockManagerPoller sendp_set_stock_exchange_name (
    in string attr_stock_exchange_name);

AMI_StockManagerPoller sendp_add_stock (
    in string symbol, in double quote);
AMI_StockManagerPoller sendp_remove_stock (
    in string symbol);


valuetype AMI_StockManagerPoller : Messaging::Poller {
    void get_stock_exchange_name (
        in unsigned long timeout,
        out string ami_return_val);
    void set_stock_exchange_name (
        in unsigned long timeout);

    void add_stock (
        in unsigned long timeout,
        out boolean ami_return_val);

    void remove_stock (
        in unsigned long timeout,
        out double quote) raises (InvalidStock);
};
```

2.1.5. Components

2.1.5.1. Component Features

attributes	denote configurable properties
supported interface	inherited in all interfaces
facets	interfaces provided to the outside
receptacles	interfaces required from the outside
sources	events produced to the outside
sinks	events consumed from the outside

2.1.5.2. Component Definition Example

```
module DiningPhilosophers {
  interface IFork {
    void pick_up () raises (ForkNotAvailable);
    void release ();
```

```
  };

  component AFork {
    provides IFork fork;
  };

  eventtype PhilosopherStatus {
      public string name;
      public PhilosopherState state;
      public boolean has_left_fork;
      public boolean has_right_fork;
  };

  component APhilosopher {

    attribute string name;

    // Receptacles for forks
    uses Fork left;
    uses Fork right;

    // Source for status
    publishes PhilosopherStatus status;
  };

  component AnObserver {
    // Sink for status
    consumes PhilosopherStatus status;
  };

  ...
};
```

2.1.5.3. Navigation Interfaces

```
module Components {
    typedef string FeatureName;
    typedef sequence<FeatureName> NameList;

    valuetype PortDescription {
        public FeatureName name;
        public CORBA::RepositoryId type_id;
    };

  ...


    valuetype FacetDescription : PortDescription {
        public Object facet_ref;
    };
    typedef sequence<FacetDescription> FacetDescriptions;

    interface Navigation {
        FacetDescriptions get_all_facets ();
        Object provide_facet (in FeatureName name) raises (InvalidName);
        FacetDescriptions get_named_facets (in NameList names) raises (InvalidNa
```

```
        ...
        };

        ...


        valuetype PublisherDescription : PortDescription {
            public SubscriberDescriptions consumers;
        };
        typedef sequence<PublisherDescription> PublisherDescriptions;

        valuetype ConsumerDescription : PortDescription {
            public EventConsumerBase consumer;
        };
        typedef sequence<ConsumerDescription> ConsumerDescriptions;

        PublisherDescriptions get_all_publishers ();
        PublisherDescriptions get_named_publishers (in NameList names) raises (Inva

        ConsumerDescriptions get_all_consumers ();
        ConsumerDescriptions get_named_consumers (in NameList names) raises (Invali

        ...
    };
```

2.1.5.4. Assembly Interfaces

```
    uses AnInterface AReceptacle;
    consumes AnEvent ASink;


    void connect_AReceptacle (in AnInterface connection)
        raises (AlreadyConnected, InvalidConnection);
    AnInterface disconnect_AReceptacle ()
        raises (NoConnection);
    AnInterface get_connection_AReceptacle ();


    AnEventConsumer get_consumer_ASink ();


    module Components {
        ...

        interface Receptacles {
            Cookie connect (in FeatureName name, in Object connection)
                raises (InvalidName, InvalidConnection, AlreadyConnected, ExceededC
            Object disconnect (in FeatureName name, in Cookie ck)
                raises (InvalidName, InvalidConnection, CookieRequired, NoConnectio
            ConnectionDescriptions get_connections (in FeatureName name)
                raises (InvalidName);

            ...
        };

        ...
```

```
        valuetype Cookie {
            private CORBA::OctetSeq cookieValue;
        };

        valuetype SubscriberDescription {
            public Cookie ck;
            public EventConsumerBase consumer;
        };
        typedef sequence<SubscriberDescription> SubscriberDescriptions;

        interface Events {
            void connect_consumer (in FeatureName emitter_name, in EventConsumerBas
                raises (InvalidName, AlreadyConnected, InvalidConnection);
            EventConsumerBase disconnect_consumer (in FeatureName source_name)
                raises (InvalidName, NoConnection);
            EventConsumerBase get_consumer (in FeatureName sink_name) raises (Inval

            Cookie subscribe (in FeatureName publisher_name, in EventConsumerBase su
                raises (InvalidName, InvalidConnection, ExceededConnectionLimit);
            EventConsumerBase unsubscribe (in FeatureName publisher_name, in Cookie
                raises (InvalidName, InvalidConnection);

            ...
        };

        ...
    };
```

2.2. Data Distribution Service (DDS)

2.2.1. Reliability Related Policies

RELIABILITY. Selects either best effort or guaranteed delivery mechanism.

BEST_EFFORT	no special mechanism to guarantee delivery
RELIABLE	guarantee delivery at transport protocol level

OWNERSHIP. Configures potential redundancy at publisher side.

SHARED	deliver messages from all writers
EXCLUSIVE	deliver messages from live writer with highest strength

OWNERSHIP_STRENGTH. Set writer strength to be used with EXCLUSIVE OWNERSHIP policy.

2.2.2. Presentation Related Policies

PRESENTATION. Determines how change messages are presented to application.

coherent access	group change messages into explicitly delimited transactions
ordered access	preserve order of change messages
access scope	scope access options

INSTANCE	access options apply at instance scope
TOPIC	access options apply at Topic object scope
GROUP	access options apply at Publisher or Subscriber object scope

DESTINATION_ORDER. Determines how to handle concurrent updates.

BY_SOURCE_TIMESTAMP	value with highest source timestamp will be visible
BY_RECEPTION_TIMESTAMP	value with highest reception timestamp will be visible

2.2.3. History Related Policies

DURABILITY. Availability of data for late joining readers.

VOLATILE	writer does not keep any history
TRANSIENT_LOCAL	history kept in writer local memory
TRANSIENT	history kept in session local memory
PERSISTENT	history kept in persistent storage

HISTORY. How much history to keep.

KEEP_LAST	keep limited history with configurable depth
KEEP_ALL	keep all history within resource limits

RESOURCE_LIMITS. What are the available resource limits.

max_samples	maximum number of samples managed across all instances
max_instances	maximum number of managed instances
max_samples_per_instance	maximum number of samples managed per single instance

2.2.4. Timing Related Policies

DEADLINE. Guarantee periodic updates to all topic instances.

- publisher can guarantee maximum update period
- subscriber can require maximum update period

LATENCY_BUDGET. Hint on available latency reserve.

TRANSPORT_PRIORITY. Hint on requested transport priority.

LIVELINESS. Configure how entity liveliness is determined.

AUTOMATIC	service tracks Entity object liveliness
MANUAL_BY_TOPIC	publisher must periodically assert liveliness per Topic object
MANUAL_BY_PARTICIPANT	publisher must periodically assert liveliness per Participant object

LIFESPAN. Message expiration time. Relies on having synchronized clock.

TIME_BASED_FILTER. Minimum separation time for incoming messages.

2.2.5. Miscellaneous Policies

USER_DATA	attaches arbitrary data to Entity objects
TOPIC_DATA	attaches arbitrary data to Topic objects

GROUP_DATA	attaches arbitrary data to Publisher and Subscriber objects
PARTITION	define a partition name for logical domain partitioning

2.3. Enterprise JavaBeans (EJB)

2.3.1. EJB Architecture

Containers. Environment providing services to enterprise application objects

- Lifecycle management (creating and deleting instances)
- Dependency management (resource and dependency injection)
- Persistence and transactions
- ...

Enterprise Beans. Enterprise application objects managed by container

Stateful session bean	an object that lives within user session scope, state survives method invocation
Stateless session bean	an object that lives within user session scope, state only within method invocation
Singleton session bean	a singleton application object
Entity bean	an object representing persistent database state
Message driven bean	an object that handles messages, state only within message handling

Scopes and Contexts.

```
public interface Context {
    public Class <? extends Annotation> getScope ();
    public <T> T get (Contextual <T> bean);
    public <T> T get (Contextual <T> bean, CreationalContext <T> creationalCont
    ...
}
```

@Dependent	instance bound to single injection point
@RequestScoped	context associated with single method invocation or request handling
@SessionScoped	context associated with an HTTP session
@ApplicationScoped	context associated with application execution
@ConversationScoped	context associated with explicitly delimited UI interaction

2.3.2. Session Objects

2.3.2.1. Stateful Session Bean Example

```
@Stateful public class ASessionBean implements ABusinessInterface {

    // Injected reference to standard session context object
    @Resource public SessionContext sessionContext;

    // Method that is called after construction or activation
```

```
    @PostConstruct @PostActivate
    public void myInitMethod () { ... }

    // Method that is called before passivation or destruction
    @PreDestroy @PrePassivate
    public void myDoneMethod () { ... }

    // Some business methods ...
    public void myMethodOne (int iArgument) { ... }
    public int myMethodTwo (Object oArgument) { ... }

    // Business method with asynchronous interface
    @Asynchronous public Future<String> myAsynchronousMethod (String sArgument)

    // Business method that removes the bean instance
    @Remove public void myRemovalMethod () { ... }

    // Interceptor method that can also be in separate interceptor class
    @AroundInvoke
    public Object myInterceptor (InvocationContext inv)
    throws Exception {
        ...
        Object result = inv.proceed ();
        ...
        return (result);
    }
}
```

- method invocations serialized by the container
- asynchronous method invocations handled by the container
- business interface can be accessed both locally and remotely
- references obtained through dependency injection or JNDI lookup

Client life cycle view.

- accessible when reference first obtained
- removed through explicit removal method

Container life cycle view.

- instance may be passivated and activated

2.3.2.2. Singleton Session Bean Example

```
@Startup @Singleton public class ASingletonBean {

    // Injected reference to JNDI resource
    @Resource (lookup = "java:comp/env/jdbc/SomeDataSource") DataSource dataSou

    // Method that is called after construction
    @PostConstruct
    public void myInitMethod () { ... }

    // Method that is called before destruction
    @PreDestroy
    public void myDoneMethod () { ... }
```

```
    // Some business methods ...
    public synchronized void myMethodOne (int iArgument) { ... }
    public synchronized int myMethodTwo (Object oArgument) { ... }
}
```

- concurrent method invocations possible

Container life cycle view.

- construction timing determined by container
- eager construction can be requested through annotation

2.3.3. Message Driven Objects

2.3.3.1. Message Driven Bean Example

```
@MessageDriven (activationConfig = {
    @ActivationConfigProperty (
        propertyName = "destinationType",
        propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty (
        propertyName = "destinationLookup",
        propertyValue = "jms/SomeQueue")
})
public class AMessageBean implements MessageListener {

    public AMessageBean () { ... }

    @Override public void onMessage (Message aMessage) { ... }

    ...
}
```

- method invocations serialized by the container
- multiple instances can be created by the container
- message delivery order between instances is not defined

2.3.4. Entity Objects

2.3.4.1. Field Based Entity Bean Class Example

```
@Entity public class AnEntity {

    // With field based access fields are persistent by default
    private int someField;
    private String someOtherField;

    // Relationships among entities must be annotated
    @OneToMany private Collection <AnotherEntity> relatedEntities;

    // Every entity must have a primary key
    @Id private long aKeyField;

    // Field that is not persistent
```

```
    @Transient private String aTransientString;

    // Version field for optimistic concurrency
    @Version private long version = 0L;

    // Obligatory constructor with no arguments
    public AnEntity () { ... }

    // Additional business methods ...
    public void myMethodOne (int iArgument) { ... }
    public int myMethodTwo (Object oArgument) { ... }
}
```

2.3.4.2. Property Based Entity Bean Class Example

```
@Entity public class AnEntity {

    // With property based access fields are not persistent themselves
    private int someTransientField;
    private String someOtherTransientField;

    // Relationships among entities must be annotated
    private Collection <AnotherEntity> relatedEntities;
    @OneToMany public Collection <AnotherEntity> getRelatedEntities () {
        return (relatedEntities);
    }
    public void setRelatedEntities (Collection <AnotherEntity> entityCollection
        relatedEntities = entityCollection;
    }

    // Getter and setter methods for primary key
    private long aKeyField;
    @Id Long getAKeyField () { return (aKeyField); }
    public void setAKeyField (Long aKeyField) { this.aKeyField = aKeyField; }

    // Obligatory constructor with no arguments
    public AnEntity () { ... }

    // Additional business methods ...
    public void myMethodOne (int iArgument) { ... }
    public int myMethodTwo (Object oArgument) { ... }
}
```

2.3.4.3. Entity Manager Interface

```
public interface EntityManager {

    void persist (Object entity);
    void refresh (Object entity);
    void remove (Object entity);

    void detach (Object entity);
    <T> T merge (T entity);

    void lock (Object entity, LockModeType lockMode);
```

```
        // Find by primary key
        <T> T find (Class <T> entityClass, Object primaryKey);

        // Find by primary key and return lazy reference
        <T> T getReference (Class <T> entityClass, Object primaryKey);

        // Clear persistence context and detach all entities
        void clear ();

        // Check whether persistence context contains managed entity
        boolean contains (Object entity);

        // Synchronize persistence context with database
        // Flush mode governs automatic synchronization
        // upon query execution or upon commit
        void flush ();
        FlushModeType getFlushMode ();
        void setFlushMode (FlushModeType flushMode);

        Query createQuery (String ejbqlString);
        Query createNamedQuery (String name);
        Query createNativeQuery (String sqlString);
        ...
    }
```

2.3.4.4. Query Interface

```
    public interface Query {

        // Execute a query that returns a result list
        List getResultList ();
        // Execute a query that returns a single result
        Object getSingleResult();
        // Execute an update query
        int executeUpdate ();

        // Methods used to fetch results step by step
        Query setMaxResults (int maxResult);
        Query setFirstResult (int startPosition);

        // Bind a parameter in a query
        Query setParameter (String name, Object value);
        Query setParameter (String name, Date value, TemporalType temporalType);
        Query setParameter (String name, Calendar value, TemporalType temporalType)
        Query setParameter (int position, Object value);
        Query setParameter (int position, Date value, TemporalType temporalType);
        Query setParameter (int position, Calendar value, TemporalType temporalType
    }
```

2.3.5. Transactions

2.3.5.1. Transaction Annotations

```
    @Stateful @TransactionManagement (BEAN) public class SomeBeanClass implements S
        @Resource javax.Transaction.UserTransaction transaction;
```

```
    public void myMethodOne () {
        transaction.begin ();
        ...
    }

    public void myMethodTwo () {
        ...
        transaction.commit ();
    }
}

@Stateless public class SomeBeanClass implements SomeBeanInterface {
    @TransactionAttribute (REQUIRED) public void myMethod () {
        ...
    }
}
```

BEAN	bean demarcated transactions
NEVER	transaction must not be active or exception is thrown
MANDATORY	transaction must be active or exception is thrown
SUPPORTS	no specific demarcation performed by the container
NOT_SUPPORTED	active transaction suspended during call
REQUIRED	active transaction used or new transaction started during call
REQUIRES_NEW	active transaction suspended and new transaction started during call

2.4. etcd

2.4.1. Etcd Data Model

Single cluster wide key value store

- Keys and values are byte arrays
- Keys are sorted lexicographically
- Store has monotonically increasing revisions
- Used to timestamp key creation and modification
- Historical revisions available until compaction
- Keys have increasing versions
- Keys can have leases

```
message KeyValue {
    bytes key = 1;
    int64 create_revision = 2;
    int64 mod_revision = 3;
    int64 version = 4;
    bytes value = 5;
    int64 lease = 6;
}
```

2.4.2. Put Request

```
rpc Put (PutRequest) returns (PutResponse) { }
```

```
message PutRequest {
    bytes key = 1;
    bytes value = 2;

    // Lease identifier or 0 for no lease
    int64 lease = 3;

    // Optionally return previous key value pair
    bool prev_kv = 4;

    // Optionally update using existing value
    bool ignore_value = 5;

    // Optionally update using existing lease
    bool ignore_lease = 6;
}

message PutResponse {
    ResponseHeader header = 1;
    KeyValue prev_kv = 2;
}
```

2.4.3. Range Request

```
rpc Range (RangeRequest) returns (RangeResponse) { }

message RangeRequest {

    enum SortOrder {
        NONE = 0;
        ASCEND = 1;
        DESCEND = 2;
    }

    enum SortTarget {
        KEY = 0;
        VERSION = 1;
        CREATE = 2;
        MOD = 3;
        VALUE = 4;
    }

    bytes key = 1;
    bytes range_end = 2;

    // Restrict number of keys returned
    int64 limit = 3;

    // Possibly query historical revision
    int64 revision = 4;

    SortOrder sort_order = 5;
    SortTarget sort_target = 6;

    // Linearizable returns cluster consensus
    // Serializable can return stale data
    bool serializable = 7;
```

```
    bool keys_only = 8;
    bool count_only = 9;

    int64 min_mod_revision = 10;
    int64 max_mod_revision = 11;

    int64 min_create_revision = 12;
    int64 max_create_revision = 13;
}

message RangeResponse {
    ResponseHeader header = 1;
    repeated KeyValue kvs = 2;
    bool more = 3;
    int64 count = 4;
}
```

- also delete range request

2.4.4. Transaction Request

```
rpc Txn (TxnRequest) returns (TxnResponse) { }

message TxnRequest {
    // List of tests to perform before transaction
    repeated Compare compare = 1;
    // List of operations to perform when all tests succeed
    repeated RequestOp success = 2;
    // List of operations to perform when any test fails
    repeated RequestOp failure = 3;
}

message Compare {

    enum CompareResult {
        EQUAL = 0;
        GREATER = 1;
        LESS = 2;
        NOT_EQUAL = 3;
    }

    enum CompareTarget {
        VERSION = 0;
        CREATE = 1;
        MOD = 2;
        VALUE = 3;
        LEASE = 4;
    }

    CompareResult result = 1;
    CompareTarget target = 2;

    bytes key = 3;
    oneof target_union {
        int64 version = 4;
        int64 create_revision = 5;
```

```
        int64 mod_revision = 6;
        bytes value = 7;
        int64 lease = 8;
    }

    // Can compare key range rather than just one key
    bytes range_end = 64;
}

message RequestOp {
    oneof request {
        RangeRequest request_range = 1;
        PutRequest request_put = 2;
        DeleteRangeRequest request_delete_range = 3;
        TxnRequest request_txn = 4;
    }
}

message TxnResponse {
    ResponseHeader header = 1;
    bool succeeded = 2;
    repeated ResponseOp responses = 3;
}
```

2.4.5. Watch Request

```
rpc Watch (stream WatchRequest) returns (stream WatchResponse) { }

message WatchRequest {
    oneof request_union {
        WatchCreateRequest create_request = 1;
        WatchCancelRequest cancel_request = 2;
        WatchProgressRequest progress_request = 3;
    }
}

message WatchCreateRequest {

    enum FilterType {
        NOPUT = 0;
        NODELETE = 1;
    }

    bytes key = 1;
    bytes range_end = 2;
    int64 start_revision = 3;

    // Request keepalive notifications
    bool progress_notify = 4;

    repeated FilterType filters = 5;

    bool prev_kv = 6;
    int64 watch_id = 7;
    bool fragment = 8;
}
```

```
message WatchResponse {
    ResponseHeader header = 1;
    int64 watch_id = 2;
    bool created = 3;
    bool canceled = 4;

    // Indicates attempt to watch already compacted revision
    int64 compact_revision = 5;

    string cancel_reason = 6;

    bool fragment = 7;

    repeated Event events = 11;
}

message Event {

    enum EventType {
        PUT = 0;
        DELETE = 1;
    }

    EventType type = 1;
    KeyValue kv = 2;
    KeyValue prev_kv = 3;
}
```

- events reported before cluster consensus

2.4.6. Lease Request

```
rpc LeaseGrant (LeaseGrantRequest) returns (LeaseGrantResponse) { }

message LeaseGrantRequest {
    // Advisory time to live in seconds
    int64 TTL = 1;
    int64 ID = 2;
}

message LeaseGrantResponse {
    ResponseHeader header = 1;
    int64 ID = 2;
    int64 TTL = 3;
    string error = 4;
}
```

- also lease revoke request
- also lease keep alive request
- also lease time to live query request

2.4.7. Lock Request

```
rpc Lock (LockRequest) returns (LockResponse) { }
rpc Unlock (UnlockRequest) returns (UnlockResponse) { }
```

```
message LockRequest {
    bytes name = 1;
    int64 lease = 2;
}

message LockResponse {
    ResponseHeader header = 1;
    bytes key = 2;
}

message UnlockRequest {
    bytes key = 1;
}

message UnlockResponse {
    ResponseHeader header = 1;
}
```

- lock `name` creates key `name/uuid`
- lock key must be checked in exclusive operations to ensure consistency

2.4.8. Leader Election Request

```
rpc Campaign (CampaignRequest) returns (CampaignResponse) { }
rpc Proclaim (ProclaimRequest) returns (ProclaimResponse) { }
rpc Leader (LeaderRequest) returns (LeaderResponse) { }
rpc Observe (LeaderRequest) returns (stream LeaderResponse) { }
rpc Resign (ResignRequest) returns (ResignResponse) { }

message CampaignRequest {
    bytes name = 1;
    int64 lease = 2;
    bytes value = 3;
}

message CampaignResponse {
    ResponseHeader header = 1;
    LeaderKey leader = 2;
}

message LeaderKey {
    bytes name = 1;
    bytes key = 2;
    int64 rev = 3;
    int64 lease = 4;
}

message ProclaimRequest {
    LeaderKey leader = 1;
    bytes value = 2;
}

message ProclaimResponse {
    ResponseHeader header = 1;
}
```

```
message LeaderRequest {
    bytes name = 1;
}

message LeaderResponse {
    ResponseHeader header = 1;
    KeyValue kv = 2;
}
```

- leader proclaims shared value
- followers observe proclaimed value

2.5. Felix

2.5.1. iPOJO Service Requirement

```
// Service reference injected into field.
@Requires private LogService log;

// Service reference injected into constructor argument.
public MyComponent (@Requires LogService log) { ... }

// Service reference injected through method invocation.
@Bind public void bindLogService(LogService log) { ... }
@Unbind public void unbindLogService(LogService log) { ... }
@Modified public void modifiedLogService(LogService log) { ... }

// Example adjusted from documentation, see references.
```

2.5.2. iPOJO Service Provision

```
// Service provision with implicitly declared interfaces.
@Component @Provides
public class FooProvider implements FooService {
    ...
}

// Service provision with explicitly declared interfaces.
@Component @Provides (specifications={FooService.class})
public class FooProvider implements FooService {
    ...

    // Public service property declaration.
    @ServiceProperty (name="foo", value="foo")
    private String aFoo;

    // Private component property declaration.
    @Property (name="bar", falue="bar")
    private String aBar;

    // Property change notification.
    @Updated public void updated (Dictionary properties) {
        ...
    }
```

```
}

// Example adjusted from documentation, see references.
```

2.5.3. iPOJO Lifecycle Management

```
// Component with requirements and lifecycle management.
@Component @Instantiate
public class FooComponent {
    @Requires private LogService log;

    @Validate private void start () {
        // Called when all instance requirements become available.
        ...
    }

    @Invalidate private void stop () {
        // Called when some instance requirement ceases being available.
        ...
    }

    // Setting controller field to false disables component instance.
    @Controller private boolean enabled;
}

// Example adjusted from documentation, see references.
```

2.6. FlatBuffers

2.6.1. Schema Language

2.6.1.1. FlatBuffers Schema Example

```
namespace SomeNamespace;

enum SomeEnum: int { One = 1, Two, Three }

struct SomeStructure {

    // Basic integer types.
    a_byte: byte;
    a_short: short;
    an_int: int;
    a_long: long;
    an_unsigned_short: ushort;
    an_unsigned_int: uint;
    an_unsigned_long: ulong;

    // Basic float types.
    a_float: float;
    a_double: double;

    // Array field only supported in structures.
    an_int_array: [int: 123];
```

```
}

table SomeTable {

    // Optional fields.
    a_byte: byte;
    an_int: int;
    a_string: string;

    // Required field of non basic types.
    a_required_string: string (required);
    a_required_enum_array: [SomeEnum] (required);
}

table AnotherTable {

    // Explicitly specified field identifiers.
    // Must form continuous range from 0.
    // Must be specified everywhere.
    an_int: int (id: 2);
    a_long: long (id: 0);
    a_string: string (id: 1);

    // Fields with default values.
    a_float: float = 0.0 (id: 3);
}

// Union types only for tables.
union SomeUnion { SomeTable, AnotherTable }

table RootTable {
    content: SomeUnion;
}

// Root type must be table.
root_type RootTable;
```

- A spectrum of basic types
- Structures with mandatory fields
- Extensible tables with optional fields

2.6.2. C++ Generated Code Basics

2.6.2.1. C++ Buffer Creation

Direct Creation.

```
FlatBufferBuilder builder;
auto another_table = CreateAnotherTableDirect (builder, 0x1234, "Hello FlatBuff
builder.Finish (another_table);
```

Incremental Creation.

```
FlatBufferBuilder builder;
auto some_string = builder.CreateString ("Hello FlatBuffers !");
```

```
auto another_table = CreateAnotherTable (builder, 0x1234, some_string, 0x123456
builder.Finish (another_table);
```

2.6.2.2. C++ Buffer Access

Buffer Verification.

```
uint8_t buffer [];
int size;
auto verifier = flatbuffers::Verifier (buffer, size);
if (VerifyAnotherTableBuffer (verifier)) {
    auto another_table = GetAnotherTable (buffer);
}
```

Buffer Access.

```
cout << another_table->an_int ();
cout << another_table->a_long ();
cout << another_table->a_string ()->str ();
```

2.6.3. Encoding

2.6.3.1. FlatBuffers Encoding

Buffer	Offset (uint32_t) to root table.
Table	Backward offset to field table (int32_t) followed by mix of scalar field data and field offsets. Field table contains own size (uint16_t), data size (uint16_t), list of field offsets in key order (uint16_t), 0 for not present.
Struct	Sequence of aligned scalar field data.
String	Character count followed by zero terminated data.
Vector	Item count followed by data
Union	Enum and offset to content.

2.7. gRPC

2.7.1. Interface Description Language

2.7.1.1. Protocol Buffers Message Specification Example

```
syntax = "proto3";

package org.example;

message SomeMessage {

    // Field identifiers reserved after message changes.
    reserved 8, 100;

    // Many integer types with specific encodings.
    int32 aMostlyPositiveInteger = 1;
    sint64 aSignedInteger = 2;
    uint64 anUnsignedInteger = 3;
```

```
    fixed32 anOftenBigUnsignedInteger = 4;
    sfixed32 anOftenBigSignedInteger = 5;

    // String always with UTF 8 encoding.
    string aString = 10;

    // Another message type.
    AnotherMessage aMessage = 111;

    // Variable length content supported.
    repeated string aStringList = 200;
    map <int32, string> aMap = 222;
}
```

- A spectrum of basic types
- Packages and nested types
- Fields can be repeated
- Fields are optional
- Explicit field identifiers for versioning

2.7.1.2. Protocol Buffer Service Specification Example

```
syntax = "proto3";

service AnInterface {
    rpc someMethod (SomeRequest) returns (SomeResponse) { }
    rpc secondMethod (SecondRequest) returns (stream SecondResponse) { }
    rpc thirdMethod (stream ThirdRequest) returns (ThirdResponse) { }
}

message SomeRequest { ... }
message SomeResponse { ... }
...
```

- Single or stream arguments
- Stream open during entire call

2.7.2. C++ Server Code Basics

2.7.2.1. C++ Server Implementation

Single Argument Method Implementation.

```
class MyService : public AnExampleService::Service {
    grpc.Status OneToOne (grpc.ServerContext *context,
        const AnExampleRequest *request, AnExampleResponse *response) {

        // Method implementation goes here ...

        return (grpc.Status::OK);
    }
    ...
}
```

Server Initialization.

```
MyService service;
grpc.ServerBuilder builder;
builder.AddListeningPort ("localhost:8888", grpc.InsecureServerCredentials ());
builder.RegisterService (&service);
std::unique_ptr<grpc.Server> server (builder.BuildAndStart ());

server->Wait ();
```

- Sync mode uses internal thread pool
- Async mode uses completion queues

2.7.3. Java Server Code Basics

2.7.3.1. Java Server Implementation

Single Argument Method Implementation.

```
class MyService extends AnExampleServiceGrpc.AnExampleServiceImplBase {
    @Override public void OneToOne (
        AnExampleRequest request,
        io.grpc.stub.StreamObserver<AnExampleResponse> responseObserver) {

        // Method implementation goes here ...

        responseObserver.onNext (response);
        responseObserver.onCompleted ();
    }
    ...
}
```

Server Initialization.

```
io.grpc.Server server = io.grpc.ServerBuilder
    .forPort (8888).addService (new MyService ()).build ().start ();

server.awaitTermination ();
```

- Uses static cached thread pool by default
- Can use provided executor
- Can use transport thread

2.7.4. Python Server Code Basics

2.7.4.1. Python Server Implementation

Single Argument Method Implementation.

```
class MyServicer (AnExampleServiceServicer):
    def OneToOne (self, request, context):

        # Method implementation goes here ...
```

```
        return response
```

Server Initialization.

```
server = grpc.server (
    futures.ThreadPoolExecutor (
        max_workers = SERVER_THREAD_COUNT))
add_AnExampleServiceServicer_to_server (MyServicer (), server)
server.add_insecure_port ("localhost:8888")
server.start ()

server.wait_for_termination ()
```

2.7.5. C++ Client Code Basics

2.7.5.1. C++ Client Implementation

Client Initialization.

```
std::shared_ptr<grpc.Channel> channel = grpc.CreateChannel (
    "localhost:8888", grpc.InsecureChannelCredentials ());
```

Single Argument Method Call.

```
grpc.ClientContext context;
AnExampleResponse response;
std::shared_ptr<AnExampleService::Stub> stub = AnExampleService::NewStub (channe
grpc.Status status = stub->OneToOne (&context, request, &response);
if (status.ok ()) {

    // Response available here ...

}
```

2.7.6. Java Client Code Basics

2.7.6.1. Java Client Implementation

Client Initialization.

```
io.grpc.ManagedChannel channel = io.grpc.ManagedChannelBuilder
    .forAddress ("localhost", 8888)
    .usePlaintext (true)
    .build ();
```

Single Argument Method Call.

```
AnExampleServiceGrpc.AnExampleServiceBlockingStub stub =
    AnExampleServiceGrpc.newBlockingStub (channel);
AnExampleResponse response = stub.oneToOne (request);

// Response available here ...
```

2.7.7. Python Client Code Basics

2.7.7.1. Python Client Implementation

Client Initialization.

```
with grpc.insecure_channel ("localhost:8888") as channel:
```

Single Argument Method Call.

```
stub = AnExampleServiceStub (channel)
response = stub.OneToOne (request)

# Response available here ...
```

2.8. Hazelcast

2.8.1. Hazelcast Architecture

Topologies.

Embedded	Node is part of client (Java)
Client Server	Nodes are separate servers with connected clients (Java, Python, C++, C# ...)
Smart Client	Connects to all server nodes and distributes requests
Single Socket Client	Connects to one server node that mediates requests

Partitioning. Partitioned data structures split between nodes

- By default 271 partitions
- Per instance configurable backup copies
- Per instance configurable synchronization with backup copies
- Read of backup copies possible with reduced consistency guarantees

Nodes can form partition groups

- Used to distribute partitions across failure domains
- Can be derived from deployment architecture in cloud

Partitioning uses consistent hash algorithm

- Smart clients can communicate with relevant nodes directly
- Non partitioned data structures can specify partition manually

2.8.2. Distributed Collections

Map	distributed hash map with possible persistency
Set	distributed hash set with possible persistency
Multi Map	a hash map variant that supports multiple values per key
Replicated Map	a hash map variant that stores all entries everywhere
Queue	distributed blocking queue
List	ordered list stored on one node
Ring Buffer	distributed circular buffer
Event Journal	distributed map update journal
Cardinality Estimator	distributed set cardinality estimator

2.8.3. IMap Interface

```
public interface IMap <K,V> extends ConcurrentMap <K,V>, BaseMap<K,V>, Iterable

    // Synchronous access methods

    V get (Object key);
    Map <K,V> getAll (Set <K> keys);

    void set (K key, V value);
    void setAll (Map <? extends K, ? extends V> map);

    V put (K key, V value);
    V putIfAbsent (K key, V value);
    void putAll (Map <? extends K, ? extends V> map);

    V replace (K key, V value);
    boolean replace (K key, V oldValue, V newValue);

    V remove (Object key);
    boolean remove (Object key, Object value);
    void removeAll (Predicate <K,V> predicate);

    void delete (Object key);
    void clear ();

    boolean containsKey (Object key);
    boolean containsValue (Object value);

    Iterator <Entry <K,V>> iterator ();
    Iterator <Entry <K,V>> iterator (int fetchSize);

    Set <K> keySet ();
    Set <K> keySet (Predicate <K,V> predicate);
    Set <K> localKeySet ();
    Set <K> localKeySet (Predicate <K,V> predicate);
    Collection <V> values ();
    Collection <V> values (Predicate <K,V> predicate);
    Set <Map.Entry <K,V>> entrySet ();
    Set <Map.Entry <K,V>> entrySet (Predicate <K,V> predicate);

    // Entries can have limited lifetime

    boolean setTtl (K key, long ttl, TimeUnit timeunit);

    void set (K key, V value, long ttl, TimeUnit ttlUnit);
    void set (K key, V value, long ttl, TimeUnit ttlUnit, long maxIdle, TimeUni

    V put (K key, V value, long ttl, TimeUnit ttlUnit);
    V put (K key, V value, long ttl, TimeUnit ttlUnit, long maxIdle, TimeUnit ma
    V putIfAbsent (K key, V value, long ttl, TimeUnit ttlUnit);
    V putIfAbsent (K key, V value, long ttl, TimeUnit ttlUnit, long maxIdle, Ti

    // Keys can be locked even if absent

    void lock (K key);
```

```
void lock (K key, long leaseTime, TimeUnit timeUnit);
boolean tryLock (K key);
boolean tryLock (K key, long time, TimeUnit timeunit);
boolean tryLock (K key, long time, TimeUnit timeunit, long leaseTime, TimeU

void unlock (K key);
void forceUnlock (K key);

boolean isLocked (K key);

boolean tryPut (K key, V value, long timeout, TimeUnit timeunit);
boolean tryRemove (K key, long timeout, TimeUnit timeunit);

// Asynchronous access methods

CompletionStage <V> getAsync (K key);

CompletionStage <Void> setAsync (K key, V value);
CompletionStage <Void> setAsync (K key, V value, long ttl, TimeUnit ttlUnit
CompletionStage <Void> setAsync (K key, V value, long ttl, TimeUnit ttlUnit,
CompletionStage <Void> setAllAsync (Map <? extends K, ? extends V> map);

CompletionStage <V> putAsync (K key, V value);
CompletionStage <V> putAsync (K key, V value, long ttl, TimeUnit ttlUnit);
CompletionStage <V> putAsync (K key, V value, long ttl, TimeUnit ttlUnit, l
CompletionStage <Void> putAllAsync (Map <? extends K, ? extends V> map);

CompletionStage <V> removeAsync (K key);

// Non transient entries can have backing store

void loadAll(boolean replaceExistingValues);
void loadAll(Set<K> keys, boolean replaceExistingValues);

boolean evict (K key);
void evictAll ();
void flush();

void putTransient (K key, V value, long ttl, TimeUnit ttlUnit);
void putTransient (K key, V value, long ttl, TimeUnit ttlUnit, long maxIdle

// Local and global state change listeners with predicate filtering are sup

UUID addLocalEntryListener (MapListener listener);
UUID addLocalEntryListener (MapListener listener, Predicate <K,V> predicate
UUID addLocalEntryListener (MapListener listener, Predicate <K,V> predicate

UUID addEntryListener (MapListener listener, boolean includeValue);
UUID addEntryListener (MapListener listener, K key, boolean includeValue);
UUID addEntryListener (MapListener listener, Predicate <K,V> predicate, boo
UUID addEntryListener (MapListener listener, Predicate <K,V> predicate, K k

boolean removeEntryListener (UUID id);

UUID addPartitionLostListener (MapPartitionLostListener listener);
boolean removePartitionLostListener (UUID id);

// Interceptors can modify or cancel operations
```

```
    String addInterceptor (MapInterceptor interceptor);
    boolean removeInterceptor (String id);

    // Entry view provides entry access statistics

    EntryView <K,V> getEntryView (K key);
    LocalMapStats getLocalMapStats ();

    // Distributed processing

    <R> R executeOnKey (K key, EntryProcessor <K,V,R> entryProcessor);
    <R> Map <K,R> executeOnKeys (Set<K> keys, EntryProcessor <K,V,R> entryProces
    <R> Map <K,R> executeOnEntries (EntryProcessor <K,V,R> entryProcessor);
    <R> Map <K,R> executeOnEntries (EntryProcessor <K,V,R> entryProcessor, Predi

    <R> Collection <R> project (Projection <? super Map.Entry<K,V>,R> projectio
    <R> Collection <R> project (Projection <? super Map.Entry<K,V>,R> projectio

    <R> R aggregate (Aggregator <? super Map.Entry <K,V>,R> aggregator);
    <R> R aggregate (Aggregator <? super Map.Entry <K,V>,R> aggregator, Predica

    <R> CompletionStage <R> submitToKey (K key, EntryProcessor <K,V,R> entryPro
    <R> CompletionStage <Map<K,R>> submitToKeys (Set<K> keys, EntryProcessor <K

    // Cache for continuous queries defined by predicates

    QueryCache <K,V> getQueryCache (String name);
    QueryCache <K,V> getQueryCache (String name, Predicate <K,V> predicate, boo
    QueryCache <K,V> getQueryCache (String name, MapListener listener, Predicat

    ...
}
```

2.8.4. Distributed Communication

Topic	publish subscribe messaging pattern implementation • configurable for sender or total ordering • totally ordered sends through topic owner

Reliable Topic	publish subscribe with backup ring buffer
	• configurable slow consumer handling
	  • `DISCARD_OLDEST` or `DISCARD_NEWEST`
	  • `BLOCK`
	  • `ERROR`

2.8.5. Distributed Coordination

Lock	distributed recursive unfair lock
Semaphore	distributed semaphore
Atomic Long	distributed counter
Atomic Reference	distributed atomic object storage (not quite reference)
Positive Negative Counter	distributed counter with relaxed consistency
Countdown Latch	distributed counter with wait for zero support

ID Generator	cluster wide unique identifier generator (for long integers) • embeds node identity to avoid communication • provides rough time ordering (k-ordering)

2.9. JGroups

2.9.1. Channels

2.9.1.1. Channel Class

```
public class JChannel implements Closeable {

    // Initialization accepts configuration options
    public JChannel ();
    public JChannel (String url);
    public JChannel (InputStream stream);

    // Join a group with a given name
    public void connect (String cluster);
    public String clusterName ();
    public void disconnect ();

    // View is the current list of members
    public View getView ();

    // Send a message to all or one group member.
    public void send (Message msg);
    public void send (Address dst, Object obj);
    public void send (Address dst, byte [] buf);
    public void send (Address dst, byte [] buf, int offset, int length);

    // Asynchronous notification about messages and membership is available
    public void setReceiver (Receiver r);
    public Receiver getReceiver ();

    ...
}
```

2.9.1.2. Receiver Interface

```
public interface Receiver {

    // Receive individual messages or batches of messages
    default void receive (Message msg) { ... }
    default void receive (MessageBatch batch) { ... }

    // Notification about membership view change
    default void viewAccepted (View new_view) { ... }

    // Notification to temporarily suspend sending messages
    default void block () { ... }
```

```
    default void unblock () { ... }

    // Group members can share state
    default void getState (OutputStream output) { ... }
    default void setState (InputStream input) { ... }
}
```

2.9.1.3. Message Classes

```
public interface Message ... {

    short BYTES_MSG   = 0,
        NIO_MSG       = 1,
        EMPTY_MSG     = 2,
        OBJ_MSG       = 3,
        LONG_MSG      = 4,
        COMPOSITE_MSG = 5,
        FRAG_MSG      = 6;

    short getType ();

    Address getDest ();
    Message setDest (Address new_dest);
    Address getSrc ();
    Message setSrc (Address new_src);

    // Headers are internal and interpreted by individual protocol modules
    Message putHeader (short id, Header hdr);
    <T extends Header> T getHeader (short id);
    Map<Short,Header> getHeaders ();

    // Flags are interpreted by individual protocol modules
    // Examples include disabling flow control or reliability
    short getFlags (boolean transient_flags);
    Message setFlag (short flag, boolean transient_flags);

    // Convenience methods on the interface
    // May not make sense for all message classes

    byte [] getArray ();
    int getOffset ();
    int getLength ();
    public Message setBuffer (byte [] b);
    Message setArray (byte [] b, int offset, int length);

    <T extends Object> T getObject ();
    Message setObject (Object obj);

    <T extends Object> T getPayload ();
    Message setPayload (Object pl);

    ...
}

public class BytesMessage ... {
    public BytesMessage (Address dest, byte [] array) { ... }
    public BytesMessage (Address dest, byte [] array, int offset, int length) {
```

```
    ...
}

public class NioMessage ... {
    // Uses java.nio.ByteBuffer that can reduce copying overhead
    public NioMessage (Address dest, ByteBuffer buf) { ... }
    public ByteBuffer getBuf () { ... }
    public NioMessage setBuf (ByteBuffer b) { ... }
    ...
}

public class ObjectMessage ... {
    public ObjectMessage(Address dest, Object obj) {
    ...
}

public class CompositeMessage ... implements Iterable<Message> {
    public CompositeMessage (Address dest, Message ... messages) { ... }
    public CompositeMessage add (Message msg) { ... }
    public <T extends Message> T get (int index) { ... }
    public Iterator<Message> iterator () { ... }
    ...
}
```

2.9.2. Building Blocks

2.9.2.1. Message Dispatcher Building Block

```
public class MessageDispatcher implements ... {

    // Message dispatcher needs channel for communication and request handler fo
    public MessageDispatcher (JChannel channel) { ... }
    public MessageDispatcher (JChannel channel, RequestHandler req_handler) { .

    // Casting sends to multiple destinations or all members when none specifie
    public <T> RspList<T> castMessage (final Collection<Address> dests, Message
    public <T> CompletableFuture<RspList<T>> castMessageWithFuture (final Colle

    // Sending sends to single destination.
    public <T> T sendMessage (Message msg, RequestOptions opts) { ... }
    public <T> CompletableFuture<T> sendMessageWithFuture (Message msg, Request

    // Request handler interface if none provided externally.
    @Override public Object handle (Message msg) { ... }
    @Override public void handle (Message request, Response response) { ... }

    ...
}

public class RequestOptions {

    // Can wait for none, one or all responses.
    public ResponseMode getMode () { ... }
    public RequestOptions setMode (ResponseMode mode) { ... }

    // Can specify response filter if response expected.
```

```
    public RspFilter getRspFilter () { ... }
    public RequestOptions setRspFilter (RspFilter filter) { ... }

    // Can specify response timeout if response expected.
    public long getTimeout () { ... }
    public RequestOptions setTimeout (long timeout) { ... }

    ...
}

public class RspList<T> extends HashMap<Address,Rsp<T>> implements Iterable<Rsp

    public int numReceived () { ... }
    public boolean isReceived (Address sender) { ... }

    // Standard get inherited.
    public T getFirst () { ... }
    public List<T> getResults () { ... }

    // Response is not expected from failed members.
    public int numSuspectedMembers () { ... }
    public List<Address> getSuspectedMembers () { ... }
    public boolean isSuspected (Address sender) { ... }

    ...
}
```

2.9.2.2. Atomic Counter Building Block

```
public class CounterService {

    // Channel stack must include COUNTER protocol.
    public CounterService (JChannel ch) { ... }
    public SyncCounter getOrCreateSyncCounter (String name, long initial_value)
    public CompletionStage<AsyncCounter> getOrCreateAsyncCounter (String name,
    public void deleteCounter (String name) { ... }
    ...
}

public interface SyncCounter extends BaseCounter {

    long get ();
    void set (long new_value);

    long addAndGet (long delta);
    long incrementAndGet ();
    long decrementAndGet ();

    long compareAndSwap (long expect, long update);
    boolean compareAndSet (long expect, long update);

    // Useful for complex updates under high contention.
    <T extends Streamable> T update (CounterFunction<T> updateFunction);
}

public interface AsyncCounter extends BaseCounter {
```

```
    CompletionStage<Long> get ();
    CompletionStage<Void> set (long new_value);

    ...
}
```

- Cluster coordinator stores and updates counter values
- Cluster coordinator can have backup coordinators
- Counter values include version also sent to clients
- Client value with latest version used to recover from coordinator failure

2.9.3. Protocol Modules

2.9.3.1. Transport Protocol Modules

UDP	uses IP multicast to deliver multicast messages
TCP	uses mesh of TCP connections, thread per connection model
TCP_NIO2	uses mesh of TCP connections, asynchronous single thread model
TUNNEL	tunnels transport to specialized router

2.9.3.2. Discovery Protocol Modules

PING	uses IP multicast over existing UDP transport
MPING	uses IP multicast over separate UDP transport
BPING	uses IP broadcast
TCPPING	uses list of member addresses
TCPGOSSIP	uses specialized router
FILE_PING	uses shared directory to keep track of members
JDBC_PING	uses shared database to keep track of members
RACKSPACE_PING	uses Rackspace Cloud File Storage
SWIFT_PING	uses Openstack Swift object storage
S3_PING	uses Amazon Simple Storage Service
DNS_PING	uses A and SRV records in DNS
PDC	caches discovered members

2.9.3.3. Merge Protocol Modules

MERGE2	group coordinator multicasts presence and membership view (3.X)
MERGE3	all members multicast presence and membership view

2.9.3.4. Failure Detection Modules

FD	uses periodic ping in logical ring
FD_ALL	uses multicast heartbeat
FD_ALL2	uses multicast heartbeat
FD_SOCK	uses TCP socket ring
FD_HOST	uses internal library method to ping hosts (4.X)
VERIFY_SUSPECT	verify suspect members additionally

2.9.3.5. Reliable Message Transmission Modules

NAKACK	uses negative acknowledgments and sequence numbering, old version (3.X)
NAKACK2	uses negative acknowledgments and sequence numbering, new version
UNICAST	uses positive acknowledgments and sequence numbering, for unicast messages
UNICAST2	uses negative acknowledgments and sequence numbering, for unicast messages (3.X)
UNICAST3	uses both positive and negative acknowledgments and sequence numbering, for unicast messages (4.X)

2.9.3.6. Miscellaneous Modules

UFC	rate limiting flow control for unicast
MFC	rate limiting flow control for multicast
FRAG	message fragmentation
FRAG2	message fragmentation (4.X)
STABLE	atomic delivery in group
BARRIER	helper for shared state transfer
SEQUENCER	totally ordered delivery through coordinator

RELAY2	bridge between multiple directly reachable clusters
RELAY3	bridge between multiple clusters with routing rules

AUTH	member authentication
ENCRYPT	message body encryption
COMPRESS	message body compression

2.10. Java Message Service (JMS)

2.10.1. Architecture

2.10.1.1. Connection Creation Example

```
// Get an initial naming context
Context initialContext = new InitialContext ();

// Look up the connection factory using
// a well known name in the initial context
ConnectionFactory connectionFactory;
connectionFactory = (ConnectionFactory) initialContext.lookup ("ConnectionFacto

// Create a connection using the factory
Connection connection;
connection = ConnectionFactory.createConnection ();

// A connection only delivers messages
// once it is explicitly started
connection.start ();
```

2.10.1.2. Session Creation Example

```
// Create a session for a connection, requesting
// no transaction support and automatic message
// acknowledgement
Session session;
session = connection.createSession (false, Session.AUTO_ACKNOWLEDGE);
```

2.10.1.3. Context Creation Example

```
// Create a context that includes a connection and a session.
// Use try with resources to close the context when done.
try (JMSContext context = connectionFactory.createContext ()) {
    // Create another context reusing the same connection.
    try (JMSContext another = context.createContext ()) {
```

```
            ...
        } catch (JMSRuntimeException ex) { ... }
    } catch (JMSRuntimeException ex) { ... }
```

2.10.2. Destinations

2.10.2.1. Destination Creation Example

```
Queue oQueue = oSession.createQueue ("SomeQueueName");
Topic oTopic = oSession.createTopic ("SomeTopicName");

Queue oTemporaryQueue = oSession.createTemporaryQueue ();
Topic oTemporaryTopic = oSession.createTemporaryTopic ();
```

2.10.3. Messages

2.10.3.1. Message Header

Set Directly By Sender.

JMSCorrelationID	correlated message identifier
JMSReplyTo	suggested reply destination
JMSType	message type understood by recipient

Set Indirectly By Sender.

JMSDestination	message recipient
JMSExpiration	message lifetime
JMSPriority	message priority
JMSDeliveryMode	PERSISTENT or NON_PERSISTENT
JMSDeliveryTime	earliest message delivery time

Set Automatically By Middleware.

JMSMessageID	unique message identifier
JMSTimestamp	message timestamp
JMSRedelivered	repeated delivery indication

2.10.3.2. Message Body Types

StreamMessage	stream of primitive types
MapMessage	set of named values
TextMessage	java.lang.String
ObjectMessage	serializable object
BytesMessage	byte array

2.10.4. Producers and Consumers

2.10.4.1. Producer And Consumer Creation Example

```
// Uses the classic API.

MessageProducer sender;
MessageConsumer recipient;

sender = session.createProducer (oQueue);
recipient = session.createConsumer (oQueue);
```

2.10.4.2. Producer And Consumer Creation Example

```
// Uses the simplified API.

// Configure sender with method chaining.
// Sender is not bound to destination here.
JMSProducer sender = context.createProducer ().
                     setDeliveryMode (PERSISTENT).
                     setDeliveryDelay (1000).
                     setTimeToLive (10000);

JMSConsumer recipient = context.createConsumer (oQueue);
```

2.10.4.3. Synchronous Message Send Example

```
// Uses the classic API.

TextMessage message;

message = session.createTextMessage ();
message.setText ("Hello");

// Always blocks until message is sent.
sender.send (message);
```

2.10.4.4. Synchronous Message Send Example

```
// Uses the simplified API.

// By default blocks until message is sent.
// Overloaded versions for all body types exist.
sender.send (oQueue, "Hello");
```

2.10.4.5. Message Receive Example

```
// Uses the classic API.

TextMessage oMessage;

oMessage = (TextMessage) recipient.receive ();
oMessage = (TextMessage) recipient.receive (1000);
```

2.10.4.6. Message Listener Example

```
// Uses the classic API.

public class SomeListener implements MessageListener {
    public void onMessage (Message message) {
        ...
    }
}

SomeListener oListener = new SomeListener ();
```

```
recipient.setMessageListener (oListener);
```

2.10.4.7. Message Receive Example

```
// Uses the simplified API.

// Template versions for all body types exist.
String body = consumer.receiveBody (String.class);
```

2.10.4.8. Message Filter Example

```
String selector;
MessageConsumer receiver;

selector = new String ("(SomeProperty = 1000)");
receiver = session.createConsumer (oQueue, selector);
```

2.10.4.9. Durable Subscriber Example

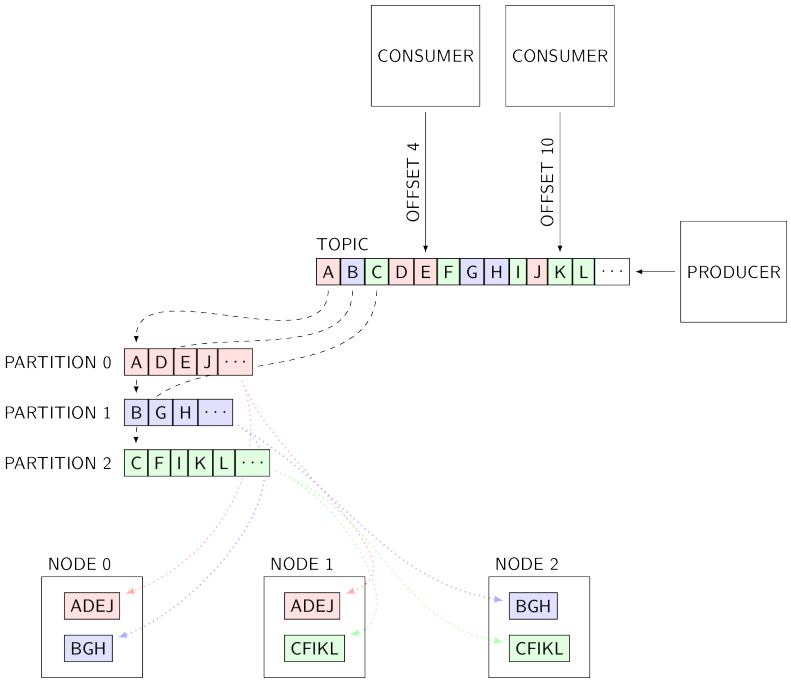
```
session.createDurableSubscriber (oTopic,"DurableSubscriberName");
```

2.10.4.10. Shared Subscriber Example

```
MessageConsumer consumer;

consumer = session.createSharedConsumer (oQueue, "SharedSubscriberName");
```

2.11. Apache Kafka

2.11.1. Kafka Architecture



Data. Data streamed in topics

- Each data record is a key value pair
- Timestamps and additional headers supported

Topics split into partitions

- Each data record stored in one partition
- Record addressed by offset within partition
- Configurable assignment of records to partitions

Brokers. Replicated broker cluster

- Each broker stores data logs of some topic partitions
- Data log retention period configurable
- Partition replication configurable

Leader follower architecture

- Topic access done on leader broker
- Leader election in case of leader failure
- Producer may require minimum number of in sync replicas

Clients. Producers

- Can batch records when so configured
- Can guarantee exactly once delivery semantics
- Can wait for confirmation from zero, one or all in sync brokers

Consumers

- Each consumer maintains own topic position
- Consumer groups split topic partitions among themselves
- Can update topic position together with output in transaction

Stream processors

2.11.2. Kafka Producer Interface

```
public class KafkaProducer <K,V> implements Producer <K,V> {
    public KafkaProducer (Properties properties) { ... }

    public Future <RecordMetadata> send (ProducerRecord <K,V> record) { ... }
    public Future <RecordMetadata> send (ProducerRecord <K,V> record, Callback

    public void flush () { ... }
    public void close () { ... }

    public void initTransactions () { ... }
    public void beginTransaction () { ... }
    public void abortTransaction () { ... }
    public void commitTransaction () { ... }

    // Introspection.
    public List <PartitionInfo> partitionsFor (String topic) { ... }

    ...
}

public class ProducerRecord <K,V> {
    public ProducerRecord (String topic, V value) { ... }
    public ProducerRecord (String topic, K key, V value) { ... }
```

```
        public ProducerRecord (
            String topic, Integer partition, K key, V value) { ... }
        public ProducerRecord (
            String topic, Integer partition, K key, V value, Iterable <Header> head
        public ProducerRecord (
            String topic, Integer partition, Long timestamp, K key, V value, Iterab

        public K key () { ... }
        public V value () { ... }
        public Headers headers () { ... }

        ...
    }

    public interface Header {
        String key ();
        byte [] value ();
    }

    public final class RecordMetadata {
        public boolean hasOffset () { ... }
        public long offset () { ... }
        public boolean hasTimestamp () { ... }
        public long timestamp () { ... }

        public String topic () { ... }
        public int partition () { ... }

        public int serializedKeySize () { ... }
        public int serializedValueSize () { ... }
    }

    public interface Callback {
        void onCompletion (RecordMetadata metadata, Exception exception);
    }
```

2.11.3. Kafka Consumer Interface

```
    public class KafkaConsumer <K,V> implements Consumer <K,V> {
        public KafkaConsumer (Properties properties) { ... }

        // Statically assigned topics and partitions.
        public void assign (Collection <TopicPartition> partitions) { ... }
        public Set <TopicPartition> assignment () { ... }

        // Specific topics with dynamically assigned partitions.
        public void subscribe (Collection <String> topics) { ... }
        public void subscribe (Collection <String> topics, ConsumerRebalanceListene

        // Regular expression topics with dynamically assigned partitions.
        public void subscribe (Pattern pattern) { ... }
        public void subscribe (Pattern pattern, ConsumerRebalanceListener listener)

        public void unsubscribe() { ... }
        public Set <String> subscription () { ... }

        // Poll for records.
```

```
        public ConsumerRecords <K,V> poll (final Duration timeout) { ... }

        // Seek and query position in topic partitions.
        public void seek (TopicPartition partition, long offset) { ... }
        public void seek (TopicPartition partition, OffsetAndMetadata offsetAndMeta
        public void seekToEnd (Collection <TopicPartition> partitions) { ... }
        public void seekToBeginning (Collection <TopicPartition> partitions) { ...

        public long position (TopicPartition partition) { ... }
        public long position (TopicPartition partition, final Duration timeout) { .

        // Set and query committed position in topic partitions.
        public void commitSync () { ... }
        public void commitSync (Duration timeout) { ... }
        public void commitSync (final Map <TopicPartition, OffsetAndMetadata> offse
        public void commitSync (final Map <TopicPartition, OffsetAndMetadata> offse
        public void commitAsync () { ... }
        public void commitAsync (OffsetCommitCallback callback) { ... }

        public OffsetAndMetadata committed (TopicPartition partition) { ... }
        public OffsetAndMetadata committed (TopicPartition partition, final Duratio

        public void pause (Collection<TopicPartition> partitions) { ... }
        public void resume (Collection<TopicPartition> partitions) { ... }
        public void close () { ... }

        // Introspection.
        public Map <String, List <PartitionInfo>> listTopics () { ... }
        public List <PartitionInfo> partitionsFor (String topic) { ... }

        ...
    }

    public class ConsumerRecord <K,V> {
        public String topic () { ... }
        public int partition () { ... }
        public long offset () { ... }
        public long timestamp () { ... }

        public K key () { ... }
        public V value () { ... }
        public Headers headers () { ... }

        public int serializedKeySize () { ... }
        public int serializedValueSize () { ... }

        ...
    }
```

2.11.4. Kafka KStream Interface

```
    public interface KStream <K,V> {

        // Filter stream by predicate.
        KStream <K,V> filter (Predicate <? super K, ? super V> predicate);
        KStream <K,V> filterNot (Predicate <? super K, ? super V> predicate);
```

```
// Replace key with new key.
<KR> KStream <KR,V> selectKey (KeyValueMapper <? super K, ? super V, ? exte

// Map entry to new entry.
<KR,VR> KStream <KR,VR> map (KeyValueMapper <
    ? super K, ? super V,
    ? extends KeyValue <? extends KR, ? extends VR>> mapper);

// Map value to new value.
<VR> KStream <K,VR> mapValues (ValueMapper <? super V, ? extends VR> mapper

// Map entry to multiple new entries.
<KR,VR> KStream <KR,VR> flatMap (KeyValueMapper <
    ? super K, ? super V,
    ? extends Iterable <? extends KeyValue <? extends KR, ? extends VR>> ma

// Map value to multiple new values.
<VR> KStream <K,VR> flatMapValues (ValueMapper <? super V, ? extends Iterab

// Print entries.
void print (Printed <K, V> printed);

// Consume or peek at entries with action.
void foreach (ForeachAction <? super K, ? super V> action);
KStream <K,V> peek (ForeachAction <? super K, ? super V> action);

// Split by predicate or merge a stream.
KStream <K,V> [] branch (Predicate <? super K, ? super V> ... predicates);
KStream <K,V> merge (KStream <K,V> stream);

// Materialize a stream into a topic.
KStream <K,V> through (String topic);
void to (String topic);

// Process a stream using a stateful processor.
<KO,VO> KStream <KO,VO> process (
    ProcessorSupplier <? super K, ? super V, KO, VO> processorSupplier,
    String... stateStoreNames);
<VO> KStream <K,VO> processValues (
    FixedKeyProcessorSupplier <? super K, ? super V, VO> processorSupplier,
    String... stateStoreNames);

// Group entries in a stream.
KGroupedStream <K,V> groupByKey ();
<KR> KGroupedStream <KR,V> groupBy (KeyValueMapper <? super K, ? super V, K

// Join stream with another stream or table on key.
// Operation on streams limited by join window.
<VO,VR> KStream <K,VR> join (
    KStream <K, VO> otherStream,
    ValueJoiner <? super V, ? super VO, ? extends VR> joiner,
    JoinWindows windows);
<VO,VR> KStream <K,VR> leftJoin (
    KStream <K, VO> otherStream,
    ValueJoiner <? super V, ? super VO, ? extends VR> joiner,
    JoinWindows windows);
<VO,VR> KStream <K,VR> outerJoin (
    KStream <K,VO> otherStream,
```

```
            ValueJoiner <? super V, ? super VO, ? extends VR> joiner,
            JoinWindows windows);
        <VT,VR> KStream <K,VR> join (
            KTable <K,VT> table,
            ValueJoiner <? super V, ? super VT, ? extends VR> joiner,
            Joined <K, V, VT> joined);
        <VT,VR> KStream <K,VR> leftJoin (
            KTable <K,VT> table,
            ValueJoiner <? super V, ? super VT, ? extends VR> joiner);

        ...
    }
```

2.11.5. Kafka KGroupedStream Interface

```
    public interface KGroupedStream <K,V> {

        KTable <K,Long> count ();
        KTable <K,V> reduce (Reducer <V> reducer);

        <VR> KTable <K,VR> aggregate (
            Initializer <VR> initializer,
            Aggregator <? super K, ? super V, VR> aggregator);

        <W extends Window> TimeWindowedKStream <K,V> windowedBy (Windows<W> windows

        ...
    }
```

2.11.6. Kafka Processor Interface

```
    public interface Processor <KIn, VIn, KOut, VOut> {

        // Lifecycle.
        default void init (final ProcessorContext <KOut, VOut> context) {}
        default void close () {}

        // Process individual records.
        void process (Record<KIn, VIn> record);
    }


    public interface ProcessorContext <KForward, VForward> extends ProcessingContex

        // Forward record to all child processors.
        <K extends KForward, V extends VForward> void forward (Record <K, V> record

        // Forward record to specified child processor.
        <K extends KForward, V extends VForward> void forward (Record <K, V> record,
    }
```

2.11.7. Kafka KeyValueStore Interface

```
    public interface ReadOnlyKeyValueStore <K, V> {
```

```
    V get (K key);

    KeyValueIterator <K, V> range (K from, K to);
    default KeyValueIterator <K, V> reverseRange (K from, K to) { ... }

    KeyValueIterator <K, V> all ();
    default KeyValueIterator <K, V> reverseAll () { ... }

    default <PS extends Serializer <P>, P> KeyValueIterator <K, V> prefixScan (

    long approximateNumEntries ();
}


public interface KeyValueStore <K, V> extends StateStore, ReadOnlyKeyValueStore

    void put (K key, V value);
    V putIfAbsent (K key, V value);
    void putAll (List <KeyValue <K, V>> entries);

    V delete (K key);
}
```

- persistent or transient store
- can be backed by topic for fault tolerance

2.12. Memcached

2.12.1. Architecture

- clients access data
  - data as key value pairs
  - key is a string (250 B limit)
  - value is array of bytes (1 MB limit)
  - client side compression supported
- servers cache data
  - standardized protocols
    - can use TCP or UDP
    - transparent binary protocol
    - text protocol limits keys and values
- servers do not know each other
- server selected by client side hashing
- least recently used strategy for eviction

2.12.2. Usage Example

```
// Initialization of memcache client.
mem = new Memcache ()
mem.add_server ("10.0.0.1:12345")
mem.add_server ("10.0.0.2:12345")
mem.add_server ("10.0.0.3:12345")


// Construct key for database query.
```

```
sql = "SELECT * FROM user WHERE name = ?"
key = "SQL:" + hash (sql) + name

// Try to fetch value from memcache.
if (defined (result = mem.get (key))) return (result)

// Fetch value from database and populate memcache otherwise.
result = execute_query (sql, name)
mem.set (key, result, lifetime)
return (result)


// Example adjusted from documentation, see references.
```

- getting and setting values of keys
- atomic setting of values for new keys
- incrementing and decrementing of integer values
- invalidating values by keys

2.13. Message Passing Interface (MPI)

2.13.1. Architecture

2.13.1.1. MPI Initialization

World Model. For programs where all processes coordinate together

```
int MPI_Init (int *argc, char ***argv);
int MPI_Init_thread (int *argc, char ***argv, int required, int *provided);
int MPI_Query_thread (int *provided);
int MPI_Is_thread_main (int *flag);
int MPI_Initialized (int *flag);
int MPI_Finalize (void);
```

MPI_THREAD_SIGNLE	single thread in this process
MPI_THREAD_FUNNELED	multiple threads but only main thread calls MPI
MPI_THREAD_SERIALIZED	multiple threads but only one MPI call at a time
MPI_THREAD_MULTIPLE	multiple threads and multiple MPI calls at a time

Session Model. For programs where processes coordinate within components

```
int MPI_Session_init (MPI_Info info, MPI_Errhandler errhandler, MPI_Session *ses
int MPI_Session_get_num_psets (MPI_Session session, MPI_Info info, int *npset_na
int MPI_Session_get_nth_pset (MPI_Session session, MPI_Info info, int n, int *ps
int MPI_Session_finalize (MPI_Session *session);
```

Process sets are administratively defined groups of processes

- arbitrary overlap possible
- intended to express shared resource scopes
- `mpi://SELF` and `mpi://WORLD` always exist

Dynamic Process Model. For explicit control over process lifecycle

```
int MPI_Comm_spawn (
    const char *command, char *argv[], int maxprocs, MPI_Info info,
    int root, MPI_Comm comm, MPI_Comm *intercomm,
    int array_of_errcodes []);

int MPI_Comm_spawn_multiple (
    int count, char *array_of_commands [], char **array_of_argv [],
    const int array_of_maxprocs [], const MPI_Info array_of_info [],
    int root, MPI_Comm comm, MPI_Comm *intercomm,
    int array_of_errcodes [])

int MPI_Comm_get_parent (MPI_Comm *parent);
```

- children have separate `MPI_COMM_WORLD`

Configuration. A set of key value pairs used to provide additional configuration

```
int MPI_Info_create (MPI_Info *info);
int MPI_Info_free (MPI_Info *info);
int MPI_Info_dup (MPI_Info info, MPI_Info *newinfo);

int MPI_Info_set (MPI_Info info, const char *key, const char *value);
int MPI_Info_get_nkeys (MPI_Info info, int *nkeys);
int MPI_Info_get_nthkey (MPI_Info info, int n, char *key);
int MPI_Info_get_string (MPI_Info info, const char *key, int *buflen, char *valu
int MPI_Info_delete (MPI_Info info, const char *key);
```

2.13.1.2. MPI Addressing

Groups. A group contains processes belonging to an application

```
MPI_GROUP_EMPTY

int MPI_Group_size (MPI_Group group, int *size);
int MPI_Group_rank (MPI_Group group, int *rank);

int MPI_Group_translate_ranks (
    MPI_Group group1, int n, const int ranks1 [],
    MPI_Group group2, int ranks2 []);

int MPI_Group_compare (MPI_Group group1, MPI_Group group2, int *result);

int MPI_Group_union (MPI_Group group1, MPI_Group group2, MPI_Group *newgroup);
int MPI_Group_difference (MPI_Group group1, MPI_Group group2, MPI_Group *newgrou
int MPI_Group_intersection (MPI_Group group1, MPI_Group group2, MPI_Group *newg

int MPI_Group_incl (MPI_Group group, int n, const int ranks [], MPI_Group *newg
int MPI_Group_excl (MPI_Group group, int n, const int ranks [], MPI_Group *newg
int MPI_Group_range_incl (MPI_Group group, int n, int ranges [][3], MPI_Group *
int MPI_Group_range_excl (MPI_Group group, int n, int ranges [][3], MPI_Group *

int MPI_Comm_group (MPI_Comm comm, MPI_Group *group);

int MPI_Group_from_session_pset (MPI_Session session, const char *pset_name, MPI

int MPI_Group_free (MPI_Group *group);
```

- created from other groups in the world model
- created from process sets in the session model
- processes addressed using rank from `0` to `size - 1`

Communicators. A communicator represents one or two groups in communication context

```
MPI_COMM_SELF
MPI_COMM_WORLD

int MPI_Comm_size (MPI_Comm comm, int *size);
int MPI_Comm_rank (MPI_Comm comm, int *rank);

int MPI_Comm_compare (MPI_Comm comm1, MPI_Comm comm2, int *result);

int MPI_Comm_dup (MPI_Comm comm, MPI_Comm *newcomm);
int MPI_Comm_dup_with_info (MPI_Comm comm, MPI_Info info, MPI_Comm *newcomm);

int MPI_Comm_create (MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm);
int MPI_Comm_create_group (MPI_Comm comm, MPI_Group group, int tag, MPI_Comm *n
int MPI_Comm_create_from_group (MPI_Group group, const char *stringtag, MPI_Inf
int MPI_Intercomm_create (
    MPI_Comm local_comm, int local_leader,
    MPI_Comm peer_comm, int remote_leader,
    int tag, MPI_Comm *newintercomm);
int MPI_Intercomm_create_from_groups (
    MPI_Group local_group, int local_leader,
    MPI_Group remote_group, int remote_leader,
    const char *stringtag, MPI_Info info, MPI_Errhandler errhandler, MPI_Comm *

int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm *newcomm);
int MPI_Comm_split_type (MPI_Comm comm, int split_type, int key, MPI_Info info,

int MPI_Intercomm_merge (MPI_Comm intercomm, int high, MPI_Comm *newintracomm);

int MPI_Comm_free (MPI_Comm *comm);

int MPI_Comm_set_info (MPI_Comm comm, MPI_Info info);
int MPI_Comm_get_info (MPI_Comm comm, MPI_Info *info_used);

int MPI_Comm_test_inter (MPI_Comm comm, int *flag);
int MPI_Comm_remote_size (MPI_Comm comm, int *size);
int MPI_Comm_remote_group (MPI_Comm comm, MPI_Group *group);
```

inter-communicator	communication within single group
intra-communicator	communication between two groups

2.13.2. Point-To-Point Communication

2.13.2.1. MPI_Send Function

```
int MPI_Send (
    const void *buf, int count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm);
int MPI_Send_c (
    const void *buf, MPI_Count count, MPI_Datatype datatype,
```

```
    int dest, int tag, MPI_Comm comm);
```

buf	address of send buffer
count	number of elements in send buffer
datatype	datatype of each send buffer element
dest	rank of destination
tag	message tag
comm	communicator

2.13.2.2. MPI_Recv Function

```
int MPI_Recv (
    void *buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm,
    MPI_Status *status);
int MPI_Recv_c (
    void *buf, MPI_Count count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm,
    MPI_Status *status);
```

buf	address of receive buffer
count	maximum number of elements in receive buffer
datatype	datatype of each receive buffer element
source	rank of source or `MPI_ANY_SOURCE`
tag	message tag or `MPI_ANY_TAG`
comm	communicator
status	status object

```
int MPI_Get_count (const MPI_Status *status, MPI_Datatype datatype, int *count)
int MPI_Get_count_c (const MPI_Status *status, MPI_Datatype datatype, MPI_Count
```

2.13.2.3. MPI_Sendrecv Function

```
int MPI_Sendrecv (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    int dest, int sendtag,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    int source, int recvtag, MPI_Comm comm,
    MPI_Status *status);
int MPI_Sendrecv_c (
    const void *sendbuf, MPI_Count sendcount, MPI_Datatype sendtype,
    int dest, int sendtag,
    void *recvbuf, MPI_Count recvcount, MPI_Datatype recvtype,
    int source, int recvtag, MPI_Comm comm,
    MPI_Status *status);
```

2.13.2.4. Point-To-Point Communication Modes

```
int MPI_Send (const void *buf, int count, MPI_Datatype datatype, int dest, int
int MPI_Bsend (const void *buf, int count, MPI_Datatype datatype, int dest, int
int MPI_Ssend (const void *buf, int count, MPI_Datatype datatype, int dest, int
int MPI_Rsend (const void *buf, int count, MPI_Datatype datatype, int dest, int
```

```
int MPI_Buffer_attach (void *buffer, int size);
int MPI_Buffer_attach_c (void *buffer, MPI_Count size);

int MPI_Buffer_detach (void *buffer_addr, int *size);
int MPI_Buffer_detach_c (void *buffer_addr, MPI_Count *size);
```

Send	may block, buffer available on return, asynchronous
BSend	does not block, uses supplied buffers, buffer available on return, asynchronous
SSend	may block, target must be receiving otherwise function fails
RSend	may block, target must be receiving otherwise undefined

```
int MPI_Isend (
    const void *buf, int count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm,
    MPI_Request *request);
int MPI_Isend_c (
    const void *buf, MPI_Count count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm,
    MPI_Request *request);

int MPI_Ibsend (...);
int MPI_Issend (...);
int MPI_Irsend (...);

int MPI_Irecv (
    void *buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm,
    MPI_Request *request);
int MPI_Irecv_c (
    void *buf, MPI_Count count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm,
    MPI_Request *request);

int MPI_Iprobe (int source, int tag, MPI_Comm comm, int *flag, MPI_Status *statu
int MPI_Improbe (int source, int tag, MPI_Comm comm, int *flag, MPI_Message *mes
int MPI_Imrecv (void *buf, int count, MPI_Datatype datatype, MPI_Message *messa
int MPI_Imrecv_c (void *buf, MPI_Count count, MPI_Datatype datatype, MPI_Messag

int MPI_Wait (MPI_Request *request, MPI_Status *status);
int MPI_Waitany (int count, MPI_Request array_of_requests [], int *index, MPI_S
int MPI_Waitall (int count, MPI_Request array_of_requests [], MPI_Status array_
int MPI_Waitsome (int incount, MPI_Request array_of_requests [], int *outcount,

int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Testany (int count, MPI_Request array_of_requests [], int *index, int *
int MPI_Testall (int count, MPI_Request array_of_requests [], int *flag, MPI_Sta
int MPI_Testsome (int incount, MPI_Request array_of_requests [], int *outcount,

int MPI_Request_free (MPI_Request *request);

int MPI_Request_get_status (MPI_Request request, int *flag, MPI_Status *status)

int MPI_Cancel (MPI_Request *request);


// Persistent communication requests serve to efficiently initiate repetitive co
```

```
int MPI_Send_init_c (
    const void *buf, MPI_Count count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm,
    MPI_Request *request);

int MPI_Recv_init_c (
    void *buf, MPI_Count count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm,
    MPI_Request *request);

int MPI_Start (MPI_Request *request);
int MPI_Startall (int count, MPI_Request array_of_requests []);


// Partitioned communication requests deliver message content in independent pa

int MPI_Psend_init (
    const void *buf, int partitions, MPI_Count count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm, MPI_Info info,
    MPI_Request *request);

int MPI_Precv_init (
    void *buf, int partitions, MPI_Count count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm, MPI_Info info,
    MPI_Request *request);

int MPI_Start (MPI_Request *request);

int MPI_Pready (int partition, MPI_Request request);
int MPI_Pready_range (int partition_low, int partition_high, MPI_Request reques
int MPI_Pready_list (int length, const int array_of_partitions [], MPI_Request

int MPI_Wait (...);
```

2.13.2.5. Data Types

Selected Basic Types.

MPI_SHORT, MPI_INT, MPI_LONG, MPI_LONG_LONG	signed integer types
MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG, MPI_UNSIGNED_LONG_LONG	unsigned integer types
MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE	floating point types
MPI_CHAR, MPI_SIGNED_CHAR, MPI_UNSIGNED_CHAR, MPI_WCHAR	character data types
MPI_INT8_T, MPI_INT16_T, MPI_INT32_T, MPI_INT64_T	exact size signed integer types
MPI_UINT8_T, MPI_UINT16_T, MPI_UINT32_T, MPI_UINT64_T	exact size unsigned integer types
MPI_BYTE	buffer with raw data
MPI_PACKED	buffer with packed data

Local Derived Types.

```
int MPI_Type_commit (MPI_Datatype *datatype);
```

```
int MPI_Type_free (MPI_Datatype *datatype);
int MPI_Type_dup (MPI_Datatype oldtype, MPI_Datatype *newtype);

int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype *newtype
int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldty

int MPI_Type_indexed (
    int count,
    const int array_of_blocklengths [],
    const int array_of_displacements [],
    MPI_Datatype oldtype,
    MPI_Datatype *newtype);

int MPI_Type_create_struct (
    int count,
    const int array_of_blocklengths [],
    const MPI_Aint array_of_displacements [],
    const MPI_Datatype array_of_types [],
    MPI_Datatype *newtype);

int MPI_Type_create_subarray (
    int ndims,
    const int array_of_sizes [], const int array_of_subsizes [],
    const int array_of_starts [],
    int order,
    MPI_Datatype oldtype,
    MPI_Datatype *newtype);
```

- elements of basic types
- offset for each element
- also versions with byte offsets
- also functions for introspecting derived types
- also functions for data import and export in canonical format

MPI_ORDER_C	row major order
MPI_ORDER_FORTRAN	column major order

Distributed Derived Types.

```
int MPI_Type_create_darray (
    int size, int rank,
    int ndims,
    const int array_of_gsizes [],
    const int array_of_distribs [], const int array_of_dargs [],
    const int array_of_psizes [],
    int order,
    MPI_Datatype oldtype,
    MPI_Datatype *newtype);
```

MPI_DISTRIBUTE_BLOCK	sequential block distribution (AAABBBCCC...)
MPI_DISTRIBUTE_CYCLIC	cyclic element distribution (ABC...ABC...ABC...)

Conversions.

- MPI types provided by communication parties must be the same
- representation conversion for portability is performed as necessary

- representation of derived types with byte offsets may not be portable

2.13.3. Collective Communication

2.13.3.1. Collective Communication Primitives

```
int MPI_Gather (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    int root, MPI_Comm comm);
int MPI_Gather_c(
    const void *sendbuf, MPI_Count sendcount, MPI_Datatype sendtype,
    void *recvbuf, MPI_Count recvcount, MPI_Datatype recvtype,
    int root, MPI_Comm comm);

int MPI_Gatherv (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts [], const int displs [], MPI_Datatype r
    int root, MPI_Comm comm);
int MPI_Gatherv_c (...);

int MPI_Igather (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    int root, MPI_Comm comm,
    MPI_Request *request);
int MPI_Igather_c (...);

int MPI_Igatherv (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts [], const int displs [],
    MPI_Datatype recvtype, int root, MPI_Comm comm,
    MPI_Request *request);
int MPI_Igatherv_c (...);

int MPI_Gather_init (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    int root, MPI_Comm comm, MPI_Info info,
    MPI_Request *request);
int MPI_Gather_init_c (...);

int MPI_Gatherv_init (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts [], const int displs [], MPI_Datatype r
    int root, MPI_Comm comm, MPI_Info info,
    MPI_Request *request);
int MPI_Gatherv_init_c (...);
```

Bcast	sender A, receivers A, A, A
Gather	senders A, B, C, receiver ABC
Scatter	sender ABC, receivers A, B, C
Allgather	senders A, B, C, receivers ABC, ABC, ABC
Alltoall	senders ABC, DEF, GHI, receivers ADG, BEH, CFI
Reduce	senders A, B, C, receiver A+B+C
Allreduce	senders A, B, C, receivers A+B+C, A+B+C, A+B+C

Reduce_scatter	senders ABC, DEF, GHI, receivers A+D+G, B+E+H, C+F+I
Scan	senders A, B, C, receivers A, A+B, A+B+C
Exscan	senders A, B, C, receivers N/A, A, A+B
Barrier	rendez vous

- semantics differ for intra and inter communication
- intra communication can use special argument for single buffer

2.13.3.2. Reduction Operations

```
MPI_SUM, MPI_PROD
MPI_MIN, MPI_MINLOC
MPI_MAX, MPI_MAXLOC
MPI_LAND, MPI_LOR, MPI_LXOR
MPI_BAND, MPI_BOR, MPI_BXOR

int MPI_Op_create (MPI_User_function *user_fn, int commute, MPI_Op *op);
int MPI_Op_free (MPI_Op *op);

typedef void MPI_User_function (void *invec, void *inoutvec, int *len, MPI_Data
```

2.13.4. Virtual Process Topologies

2.13.4.1. Virtual Topology Creation

```
int MPI_Cart_create (
    MPI_Comm comm_old,
    int ndims, const int dims [], const int periods [],
    int reorder, MPI_Comm *comm_cart);

int MPI_Cart_sub (MPI_Comm comm, const int remain_dims [], MPI_Comm *newcomm);

int MPI_Graph_create (
    MPI_Comm comm_old,
    int nnodes, const int index [], const int edges [],
    int reorder, MPI_Comm *comm_graph);

int MPI_Dist_graph_create_adjacent (
    MPI_Comm comm_old,
    int indegree, const int sources [], const int sourceweights [],
    int outdegree, const int destinations [], const int destweights [],
    MPI_Info info, int reorder, MPI_Comm *comm_dist_graph);

int MPI_Dist_graph_create (
    MPI_Comm comm_old,
    int n, const int sources [], const int degrees [], const int destinations [
    MPI_Info info, int reorder, MPI_Comm *comm_dist_graph);
```

cartesian	cartesian grid with optionally periodic dimensions
centralized graph	centralized graph definition with node degrees and flattened edge list
adjacent distributed graph	distributed graph definition which specifies incoming and outgoing edges at each node

general distributed graph	distributed graph definition which specifies arbitrary subset of edges at each node

2.13.4.2. Virtual Topology Addressing

```
int MPI_Cart_rank (MPI_Comm comm, const int coords [], int *rank);
int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int coords []);

int MPI_Cart_shift (MPI_Comm comm, int direction, int disp, int *rank_source, i

int MPI_Graph_neighbors (MPI_Comm comm, int rank, int maxneighbors, int neighbo

int MPI_Dist_graph_neighbors (
    MPI_Comm comm,
    int maxindegree, int sources [], int sourceweights [],
    int maxoutdegree, int destinations [], int destweights []);
```

2.13.4.3. Virtual Topology Communication

```
int MPI_Neighbor_allgather (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm);
int MPI_Neighbor_allgatherv (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts [], const int displs [], MPI_Datatype r
    MPI_Comm comm);

int MPI_Neighbor_alltoall (
    const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm);
int MPI_Neighbor_alltoallv (
    const void *sendbuf, const int sendcounts [], const int sdispls [], MPI_Dat
    void *recvbuf, const int recvcounts [], const int rdispls [], MPI_Datatype
    MPI_Comm comm);

int MPI_Neighbor_alltoallw (
    const void *sendbuf, const int sendcounts [], const MPI_Aint sdispls [], co
    void *recvbuf, const int recvcounts [], const MPI_Aint rdispls [], const MP
    MPI_Comm comm);
```

2.13.5. Remote Memory Access

2.13.5.1. Memory Window Initialization

```
int MPI_Win_create (
    void *base, MPI_Aint size, int disp_unit,
    MPI_Info info, MPI_Comm comm, MPI_Win *win);
int MPI_Win_create_c (
    void *base, MPI_Aint size, MPI_Aint disp_unit,
    MPI_Info info, MPI_Comm comm, MPI_Win *win);
```

```
int MPI_Win_allocate (
    MPI_Aint size, int disp_unit,
    MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win);
int MPI_Win_allocate_c (
    MPI_Aint size, MPI_Aint disp_unit,
    MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win);

int MPI_Win_allocate_shared (
    MPI_Aint size, int disp_unit,
    MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win);
int MPI_Win_allocate_shared_c (
    MPI_Aint size, MPI_Aint disp_unit,
    MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win);

int MPI_Win_create_dynamic (MPI_Info info, MPI_Comm comm, MPI_Win *win);
int MPI_Win_attach (MPI_Win win, void *base, MPI_Aint size);
int MPI_Win_detach (MPI_Win win, const void *base);

int MPI_Win_free (MPI_Win *win);
```

disp_unit unit size used in scaling remote offsets

- shared windows only possible when hardware architecture permits that
- dynamic windows permit dynamic attaching and detaching of memory with given size

2.13.5.2. Accessing Remote Memory

```
int MPI_Put (
    void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
    int target_rank, int target_disp, int target_count, MPI_Datatype target_data
    MPI_Win win);

int MPI_Get (
    void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
    int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target
    MPI_Win win);

int MPI_Accumulate (
    const void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
    int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target
    MPI_Op op,
    MPI_Win win);

int MPI_Get_accumulate (
    const void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
    void *result_addr, int result_count, MPI_Datatype result_datatype,
    int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target
    MPI_Op op,
    MPI_Win win);

int MPI_Fetch_and_op (
    const void *origin_addr, void *result_addr, MPI_Datatype datatype,
    int target_rank, MPI_Aint target_disp,
    MPI_Op op,
    MPI_Win win);

int MPI_Compare_and_swap (
```

```
    const void *origin_addr, const void *compare_addr, void *result_addr, MPI_Da
    int target_rank, MPI_Aint target_disp,
    MPI_Win win);
```

- local access requires explicit synchronization

Passive Target Synchronization.

```
int MPI_Win_flush (int rank, MPI_Win win);
int MPI_Win_flush_all (MPI_Win win);
int MPI_Win_flush_local (int rank, MPI_Win win);
int MPI_Win_flush_local_all (MPI_Win win);

int MPI_Win_lock (int lock_type, int rank, int assert, MPI_Win win);
int MPI_Win_lock_all (int assert, MPI_Win win);
int MPI_Win_unlock (int rank, MPI_Win win);
int MPI_Win_unlock_all (MPI_Win win);
```

Active Target Synchronization.

```
int MPI_Win_fence (int assert, MPI_Win win);

int MPI_Win_start (MPI_Group group, int assert, MPI_Win win);
int MPI_Win_complete (MPI_Win win);

int MPI_Win_post (MPI_Group group, int assert, MPI_Win win);
int MPI_Win_wait (MPI_Win win);
```

- assert parameter specifies application hints for optimization
- starting and completing delineates epoch of remote window access
- posting and waiting delineates epoch of exposure to remote window access

2.14. Open Services Gateway Initiative (OSGi)

2.14.1. Bundles

2.14.1.1. OSGi Bundle States

```
interface Bundle {
    // Bundle state constants
    int UNINSTALLED = 0x00000001;
    int INSTALLED = 0x00000002;
    int RESOLVED = 0x00000004;
    int STARTING = 0x00000008;
    int STOPPING = 0x00000010;
    int ACTIVE = 0x00000020;

    int getState ();

    ...
}
```

2.14.1.2. OSGi Bundle Activator Interface

```
interface BundleActivator {
    void start (BundleContext context) throws Exception;
    void stop (BundleContext context) throws Exception;
}
```

2.14.1.3. OSGi Bundle Context Interface (Bundle Related)

```
interface BundleContext {
    // Access to framework properties
    String getProperty (String key);

    // Access to objects representing bundles
    Bundle getBundle ();
    Bundle getBundle (long id);
    Bundle getBundle (String location);
    Bundle [] getBundles ();

    // Support for bundle management
    Bundle installBundle (String location, InputStream input) throws BundleExce
    Bundle installBundle (String location) throws BundleException;

    // Support for bundle lifecycle notifications
    void addBundleListener (BundleListener listener);
    void removeBundleListener (BundleListener listener);

    // Support for framework event notifications
    void addFrameworkListener (FrameworkListener listener);
    void removeFrameworkListener (FrameworkListener listener);

    // Support for persistent storage
    File getDataFile (String filename);

    ...
}
```

2.14.2. Services

2.14.2.1. OSGi Bundle Context Interface (Service Related)

```
interface BundleContext {

    ...

    // Support for service management
    ServiceRegistration registerService (String [] clazzes, Object service, Dic
    ServiceRegistration registerService (String clazz, Object service, Dictiona

    Filter createFilter (String filter) throws InvalidSyntaxException;
    ServiceReference [] getServiceReferences (String clazz, String filter) thro
    ServiceReference [] getAllServiceReferences (String clazz, String filter) t

    ServiceReference getServiceReference (String clazz);
    Object getService (ServiceReference reference);
    boolean ungetService (ServiceReference reference);
```

```
    // Support for service lifecycle notifications
    void addServiceListener (ServiceListener listener, String filter) throws Inv
    void addServiceListener (ServiceListener listener);
    void removeServiceListener (ServiceListener listener);
}
```

2.15. Protocol Buffers (protobuf)

2.15.1. Message Description Language

2.15.1.1. Protocol Buffers Message Specification Example

```
edition = "2023";

// File level options supported.
option optimize_for = SPEED;

message SomeMessage {

    // Field identifiers reserved after message changes.
    reserved 8, 100;

    // Many integer types with specific encodings.
    int32 aMostlyPositiveInteger = 1;
    sint64 aSignedInteger = 2;
    uint64 anUnsignedInteger = 3;
    fixed32 anOftenBigUnsignedInteger = 4;
    sfixed32 anOftenBigSignedInteger = 5;

    // String always with UTF 8 encoding.
    string aString = 10;

    // Another message type.
    AnotherMessage aMessage = 111;

    // Variable length content supported.
    repeated string aStringList = 333;
    map <int32, string> aMap = 444;

    // Field level options supported.
    int32 aDeprecatedInteger = 666 [deprecated = true];
    float aFloatWithoutPresenceTracking = 222 [features.field_presence = IMPLIC

    // Extension field range.
    extensions 1234 to 5678;
}

extend SomeMessage {
    // Extension field in extension field range.
    int32 anExtensionField = 1234;
}
```

- A spectrum of basic types
- Packages and nested types
- Fields can be repeated
- Fields have presence tracked unless disabled

- Explicit field identifiers for versioning
- Options tune code generation
- Extensions reserve fields

2.15.1.2. Protocol Buffers Primitive Field Types

Integer Types.

(s)fixed(32|64)	Integers with fixed length encoding
(u)int(32|64)	Integers with variable length encoding
sint(32|64)	Integers with sign optimized variable length encoding

Floating Poing Types.

float	IEEE 754 32 bit float
double	IEEE 754 64 bit float

Additional Primitive Types.

bool	Boolean
bytes	Arbitrary sequence of bytes
string	Arbitrary sequence of UTF-8 characters

2.15.1.3. Protocol Buffers More Field Types

Oneof Type.

```
message AnExampleMessage {
    oneof some_oneof_field {
        int32 some_integer = 1;
        string some_string = 2;
    }
}
```

- Assigning one field clears others

Enum Type.

```
enum AnEnum {
    INITIAL = 0;
    RED = 1;
    BLUE = 2;
    GREEN = 3;
    WHATEVER = 8;
}
```

- Must include zero

Any Type.

```
import "google/protobuf/any.proto";
message AnExampleMessage {
    repeated google.protobuf.Any whatever = 8;
}
```

- Internally a type identifier and a value
- Type identifier is URI string

- Value is byte buffer

Map Type.

```
message AnExampleMessage {
    map<int32, string> keywords = 8;
}
```

2.15.2. C++ Generated Code Basics

2.15.2.1. C++ Message Manipulation

Construction.

```
AnExampleMessage message;
AnExampleMessage message (another_message);
message.CopyFrom (another_message);
```

Singular Fields.

```
cout << message.some_integer ();
message.set_some_integer (1234);
if (message.has_optional_integer ()) {
    message.clear_optional_integer ();
}
```

Repeated Fields.

```
int size = messages.messages_size ();
const AnExampleMessage &message = messages.messages (1234);
AnExampleMessage *message = messages.mutable_messages (1234);
AnExampleMessage *message = messages.add_messages ();
```

Byte Array Serialization.

```
char buffer [BUFFER_SIZE];
message.SerializeToArray (buffer, sizeof (buffer));
message.ParseFromArray (buffer, sizeof (buffer));
```

Standard Stream Serialization.

```
message.SerializeToOstream (&stream);
message.ParseFromIstream (&stream);
```

2.15.3. Java Generated Code Basics

2.15.3.1. Java Message Manipulation

Construction.

```
AnExampleMessage.Builder messageBuilder;
messageBuilder = AnExampleMessage.newBuilder ();
```

```
messageBuilder = AnExampleMessage.newBuilder (another_message);
AnExampleMessage message = messageBulder.build ();
```

Singular Fields.

```
System.out.println (message.getSomeInteger ());
messageBuilder.setSomeInteger (1234);
if (message.hasOptionalInteger ()) {
    messageBuilder = message.toBuilder ();
    messageBuilder.clearOptionalInteger ();
}
```

Repeated Fields.

```
int size = messages.getMessagesCount ();
AnExampleMessage message = messages.getMessages (1234);
List<AnExampleMessage> messageList = messages.getMessagesList ();
messagesBuilder.addMessages (messageBuilder);
messagesBuilder.addMessages (message);
```

Byte Array Serialization.

```
byte [] buffer = message.toByteArray ();
try {
    AnExampleMessage message = AnExampleMessage.parseFrom (buffer);
} catch (InvalidProtocolBufferException e) {
    System.out.println (e);
}
```

Standard Stream Serialization.

```
message.writeTo (stream);
AnExampleMessage message = AnExampleMessage.parseFrom (stream);
```

2.15.4. Python Generated Code Basics

2.15.4.1. Python Message Manipulation

Construction.

```
message = AnExampleMessage ()
message.CopyFrom (another_message)
```

Singular Fields.

```
print (message.some_integer)
message.some_integer = 1234
if message.HasField ('optional_integer'):
    message.ClearField ('optional_integer')
```

Repeated Fields.

```
size = len (messages.messages)
```

```
message = messages.messages [1234]
message = messages.messages.add ()
```

Byte Array Serialization.

```
buffer = message.SerializeToString ()
message.ParseFromString (buffer)
message = AnExampleMessage.FromString (buffer)
```

Standard Stream Serialization.

```
file.write (message.SerializeToString ())
message.ParseFromString (file.read ())
AnExampleMessage.FromString (file.read ())
```

2.16. Redis

2.16.1. Data Model

2.16.1.1. Redis Data Model

Databases. A server can host multiple databases

- Identified by sequential numbers starting from zero
- Each database is a key value store
- Keys are binary safe strings
- Keys can have expiration time
- Values are typed

Value Types. Binary safe string

- Can also be interpreted as an integer or a float or a bitmap or a bitfield

List

- Ordered list of binary safe strings
- Atomic element removal from both ends

Set

- Set and sorted set of binary safe strings
- Sorted set keeps float score with each element

Hash

- A key value map of binary safe strings
- Keys can have expiration time

Stream

- A stream of key value maps of binary safe strings
- Individual consumers query entries by timestamp
- Consumer groups can cooperate in processing

Hyper Log Log estimator

- Opaque type for estimating set cardinality

2.16.2. Database

2.16.2.1. Database Selection Commands

SELECT	select database to use
SWAPDB	swap databases
DBSIZE	get number of keys in database
MOVE	move existing key to another database
FLUSHALL	delete all keys from all databases
FLUSHDB	delete all keys from current database

2.16.2.2. General Data Access Commands

SCAN	iterate over existign keys
KEYS	list keys matching glob
RANDOMKEY	get random existing key (but not value)
EXISTS	test existence of a key
COPY	copy existing value to another key
RENAME	move existing value to another key
RENAMENX	... if target does not exist
DEL	delete a key
UNLINK	... with asynchronous memory reclaim
EXPIRE	set relative key expiration time
PEXPIRE	... in milliseconds
EXPIREAT	set absolute key expiration time
PEXPIREAT	... in milliseconds
TTL	get key expiration time
PTTL	... in milliseconds
PERSIST	remove key expiration time
DUMP	serialize value with checksum
RESTORE	... and restore serialized value
TOUCH	set last access time for eviction policies
TYPE	get key type (string, list, hash, zset, set, stream)

2.16.2.3. String Type Access Commands

GET	get the value associated with a key
GETRANGE	... or only some characters
GETDEL	... and delete the key
GETEX	... and set key expiration time
SET	set the value associated with a key
SETRANGE	... or only some characters
SETNX	... if target does not exist
SETEX	... and set the key expiration time
PSETEX	... in milliseconds
GETSET	... and return the previous value
MGET	get values for multiple keys
MSET	set values for multiple keys
MSETNX	... if targets do not exist
APPEND	append new value to the existing value
STRLEN	get the length of the existing value
GETBIT	get single bit

SETBIT	set single bit
BITCOUNT	count bits that are set
BITPOS	find first bit that is set or reset
BITOP	perform logical operation between multiple keys
BITFIELD	perform get or set or inc on subset of bits

INCR	interpret string as integer and increment
INCRBY	... by arbitrary value
DECR	interpret string as integer and decrement
DECRBY	... by arbitrary value
INCRBYFLOAT	interpret string as float and increment or decrement

2.16.2.4. Hash Type Access Commands

HSCAN	iterate over existing fields
HKEYS	list existing fields
HVALS	get existing values
HRANDFIELD	get random existing field (and value)
HEXISTS	test existence of a field
HLEN	get the number of existing fields
HDEL	delete a field

HGET	get the value of a field
HSET	set the value of a field
HSETNX	... if target does not exist
HGETALL	get all fields and values
HMGET	get multiple fields
HMSET	set multiple fields

HSTRLEN	get the length of a field value
HINCRBY	interpret field as integer and increment or decrement
HINCRBYFLOAT	interpret field as float and increment or decrement

2.16.3. Publish Subscribe

2.16.3.1. Publish Subscribe Commands

SUBSCRIBE	subscribe to messages from given channels
PSUBSCRIBE	... with channels given by glob
UNSUBSCRIBE	unsubscribe from given channels
PUNSUBSCRIBE	... with channels given by glob

PUBLISH	publish a message on a given channel

PUBSUB CHANNELS	list channels with subscribers

- subscription restricts connection commands to pubsub
- publishing does not require prior channel connection
- pattern subscription therefore matches continuously
- key update notifications available on channels derived from key names

2.16.4. Transactional Command Execution

2.16.4.1. Transactional Command Execution

MULTI	begin a transaction
EXEC	commit a transaction

DISCARD	abort a transaction
WATCH	changes to watched keys label transaction fail only
UNWATCH	discard watched keys

- results of all operations are collected during the transaction
- individual operation failures do not terminate the transaction

2.16.5. Scripting

2.16.5.1. Scripting

EVAL	execute LUA code with arguments
EVALSHA	... or execute LUA code from cache
SCRIPT LOAD	explicitly load LUA code to cache
SCRIPT EXISTS	check presence of LUA code in cache
SCRIPT FLUSH	drop LUA code cache content

- script execution is atomic
- long running read only scripts can be terminated after timeout
- long running read write scripts cannot be terminated other than by shutdown

2.17. Java Remote Method Invocation (RMI)

2.17.1. Interface

2.17.1.1. Remotely Accessible Type Example

```
public interface Example extends Remote {
    void printString (String text) throws RemoteException;
}
```

- Inheritance used to request passing by reference
- Serializable arguments
- Remote exception

2.17.2. Implementation

2.17.2.1. Remotely Accessible Object Example

```
public class ExampleImpl extends UnicastRemoteObject implements Example {
    public ExampleImpl () throws RemoteException { }
    public void printString (String text) { System.out.println (text); }
}
```

- Interface used to mark remotely accessible object
- Inheritance used to export the instance
- Constructor can return exception

2.17.2.2. exportObject Methods

```
static Remote exportObject (Remote obj, int port)
static boolean unexportObject (Remote obj, boolean force)
```

2.17.3. Lifecycle

2.17.3.1. Unreferenced Interface

```
public interface Unreferenced {
    void unreferenced ();
}
```

- Called some time after no client holds reference to remote object

2.17.4. Naming

2.17.4.1. Naming Interface

```
class java.rmi.Naming {
    static void bind (String name, Remote obj);
    static void rebind (String name, Remote obj);
    static void unbind (String name);
    static Remote lookup (String name);
    static String [] list (String name);
}
```

2.17.4.2. Naming Example

Server Side Registration.

```
ExampleImpl obj = new ExampleImpl ();
Naming.rebind ("//localhost/Example", obj);
```

Client Side Lookup.

```
Example object = (Example) Naming.lookup ("//localhost/Example");
object.printString ("Hello RMI !");
```

2.18. Sun RPC

2.18.1. Interface Definition Example

```
const MNTPATHLEN = 1024;    /* maximum bytes in a pathname argument */
const MNTNAMLEN = 255;      /* maximum bytes in a name argument */
const FHSIZE = 32;         /* size in bytes of a file handle */

typedef opaque fhandle [FHSIZE];
typedef string name <MNTNAMLEN>;
typedef string dirpath <MNTPATHLEN>;

union fhstatus switch (unsigned fhs_status) {
    case 0:
        fhandle fhs_fhandle;
    default:
        void;
};
```

```
typedef struct mountbody *mountlist;
struct mountbody {
    name ml_hostname;
    dirpath ml_directory;
    mountlist ml_next;
};

typedef struct groupnode *groups;
struct groupnode {
    name gr_name;
    groups gr_next;
};

typedef struct exportnode *exports;
struct exportnode {
    dirpath ex_dir;
    groups ex_groups;
    exports ex_next;
};

program MOUNTPROG {
    version MOUNTVERS {
        void  MOUNTPROC_NULL (void) = 0;
        fhstatus  MOUNTPROC_MNT (dirpath) = 1;
        mountlist  MOUNTPROC_DUMP (void) = 2;
        void MOUNTPROC_UMNT (dirpath) = 3;
        void MOUNTPROC_UMNTALL (void) = 4;
        exports MOUNTPROC_EXPORT (void) = 5;
        exports MOUNTPROC_EXPORTALL (void) = 6;
    } = 1;
} = 100005;
```

2.18.2. Portmapper Services Example

```
> rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100011    1   udp    892  rquotad
    100011    2   udp    892  rquotad
    100011    1   tcp    895  rquotad
    100011    2   tcp    895  rquotad
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100021    1   udp  39968  nlockmgr
    100021    3   udp  39968  nlockmgr
    100021    4   udp  39968  nlockmgr
    100005    1   udp  39969  mountd
    100005    1   tcp  45529  mountd
    100005    2   udp  39969  mountd
    100005    2   tcp  45529  mountd
    100005    3   udp  39969  mountd
    100005    3   tcp  45529  mountd
    100024    1   udp  39970  status
    100024    1   tcp  45530  status
    391002    2   tcp  45533  sgi_fam
```

2.19. Apache Thrift

2.19.1. Interface Description Language

2.19.1.1. Thrift Interface Specification Example

```
namespace cpp org.example
namespace java org.example

enum AnEnum {
    ONE = 1,
    TWO = 2,
    THREE = 3
}

struct SomeMessage {
    1: bool aBooleanField,
    2: i8 aByteField,
    3: i16 aShortIntField,
    4: i32 aNormalIntField,
    5: i64 aLongIntField,

    10: double aDoubleField,

    20: string aStringField,
    22: binary aBinaryField,

    // Fields can have default values.
    100: AnEnum anEnumFieldWithDefault = AnEnum.THREE,

    // Fields can be optional.
    200: optional i16 anOptionalIntField
}

// Exceptions are structures too.
exception SomeException {
    1: list<string> aStringList,
    2: set<i8> aByteSet,
    3: map<i16, string> aMap
}

service AnInterface {
    void ping (),
    bool aMethod (1: i16 argShort, 2: i64 argLong),
    oneway void aOnewayMethod (1: SomeMessage message)
}

service AnotherInterface extends AnInterface {
    void yetAnotherMethod (1: SomeMessage message) throws (1: SomeException ex)
}
```

- Namespaces per language
- A spectrum of basic types
- Containers for other types
- Fields can be optional
- Explicit field and argument identifiers for versioning

- Methods can be oneway
- Methods can throw exceptions

2.19.2. C++ Server Code Basics

2.19.2.1. C++ Server Implementation

Method Implementation.

```
class ExampleHandler : virtual public ExampleIf {
    void printString (const std::string &text) override {

        // Method implementation goes here ...

    }
    ...
}
```

Server Initialization.

```
// Handler is the user defined implementation.
std::shared_ptr<ExampleHandler> handler (new ExampleHandler ());

// Processor is responsible for decoding function arguments and invoking the ha
std::shared_ptr<ExampleProcessor> processor (new ExampleProcessor (handler));

// Transport provides reading and writing of byte buffers.
std::shared_ptr<TServerTransport> transport (new TServerSocket (SERVER_PORT));

// Buffered transport is a wrapper for another transport object.
std::shared_ptr<TTransportFactory> transport_factory (new TBufferedTransportFac

// Protocol provides reading and writing of individual types on top of transpor
std::shared_ptr<TProtocolFactory> protocol_factory (new TBinaryProtocolFactory

// New connections use their own transport and protocol instances hence the fac
std::shared_ptr<TServer> server (new TSimpleServer (processor, transport, trans

server->serve ();
```

- Public `read` and `write` methods on generated transport types
- Multiple transports available
  - Plain socket
  - Socket with SSL
  - Socket with HTTP
  - Socket with WebSocket
  - Wrapper for zlib compression
  - File and pipe
  - Memory buffer
- Multiple protocols available
  - JSON
  - Simple binary encoding
  - Compact binary encoding
  - Wrapper for serving multiple services
- Multiple servers available
  - Single main thread

- Thread per connection
- Thread pool with fixed size
- Thread pool with fixed size and dedicated dispatcher

2.19.3. C++ Client Code Basics

2.19.3.1. C++ Client Implementation

Client Initialization.

```
// Transport provides reading and writing of byte buffers.
std::shared_ptr<TTransport> socket (new TSocket (SERVER_ADDR, SERVER_PORT));

// Buffered transport is a wrapper for another transport object.
std::shared_ptr<TTransport> transport (new TBufferedTransport (socket));

// Protocol provides reading and writing of individual types on top of transpor
std::shared_ptr<TProtocol> protocol (new TBinaryProtocol (transport));
```

Method Call.

```
std::shared_ptr<ExampleClient> client (new ExampleClient (protocol));
client->printString ("Hello from Thrift in C++ !");
```

2.20. Web Services

2.20.1. SOAP

2.20.1.1. Message Example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
               SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Header with additional information -->
    <SOAP:Header>
        <wscoor:CoordinationContext
            xmlns:wscoor="http://schemas.xmlsoap.org/ws/2003/09/wscoor"
            SOAP:mustUnderstand="true">
            <wscoor:Identifier>
                http://example.com/context/1234
            </wscoor:Identifier>
        </wscoor:CoordinationContext>
    </SOAP:Header>

    <!-- Body with message content -->
    <SOAP:Body>
        <m:aMethodRequest xmlns:m="http://example.com/soap.wsdl">
            <aNumber xsi:type="xsd:int">42</aNumber>
        </m:aMethodRequest>
    </SOAP:Body>

</SOAP:Envelope>
```

2.20.2. WSDL

2.20.2.1. Service Example

```
<?xml version="1.0"?>
<definitions name="StockQuote"
             targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

    <!-- Types used in communication -->
    <types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
                xmlns="http://www.w3.org/2000/10/XMLSchema">
            <element name="TradePriceRequest">
                <complexType>
                    <all>
                        <element name="tickerSymbol" type="string"/>
                    </all>
                </complexType>
            </element>
            <element name="TradePriceReply">
                 <complexType>
                     <all>
                         <element name="price" type="float"/>
                     </all>
                 </complexType>
              </element>
         </schema>
      </types>

    <!-- Messages exchanged in communication -->
    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd:TradePriceRequest"/>
    </message>
    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd:TradePriceReply"/>
    </message>

    <!-- Ports available in communication -->
    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <!-- Bindings used in communication -->
    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding
            style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
```

```
            <input><soap:body use="literal"/></input>
            <output><soap:body use="literal"/></output>
        </operation>
    </binding>

    <!-- Service -->
    <service name="StockQuoteService">
        <documentation>Stock quoter service.</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
            <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>

<!-- Example adjusted from WSDL 1.1 Specification. -->
```

2.20.3. BPEL

2.20.3.1. BPEL Example

```
<process name="anExampleProcess">

    <!-- Partners of the example process -->
    <partnerLinks>
        <partnerLink name="client"
            partnerLinkType="aClientPort"
            myRole="aProviderRole"/>
        <partnerLink name="serverOne"
            partnerLinkType="aServerPort"
            myRole="aClientRole"
            partnerRole="aServerRole"/>
        <partnerLink name="serverTwo"
            partnerLinkType="aServerPort"
            myRole="aClientRole"
            partnerRole="aServerRole"/>
    </partnerLinks>

    <!-- Internal variables -->
    <variables>
        <variable name="clientRequest" messageType="RequestMessage"/>
        <variable name="serverOneResponse" messageType="ResponseMessage"/>
        <variable name="serverTwoResponse" messageType="ResponseMessage"/>
        <variable name="providerResponse" messageType="ResponseMessage"/>
    </variables>

    <!-- Process definition -->
    <sequence>
        <!-- Get the request from the client -->
        <receive partnerLink="client"
            portType="aClientPort"
            operation="GetOffer"
            variable="clientRequest"
            createInstance="yes"/>

        <!-- Forward the request to both servers -->
```

```
            <flow>
                <invoke partnerLink="serverOne"
                    portType="aServerPort"
                    operation="GetOffer"
                    inputVariable="clientRequest"
                    outputVariable="serverOneResponse"/>
                <invoke partnerLink="serverTwo"
                    ... />
            </flow>

            <!-- Create response from cheapest offer -->
            <switch>
                <case condition="bpws:getVariableData ('serverOneResponse','price')
                                 <
                                 bpws:getVariableData ('serverTwoResponse','price')
                    <assign>
                        <copy>
                            <from variable="serverOneResponse"/>
                            <to variable="providerResponse"/>
                        </copy>
                    </assign>
                </case>
                <otherwise>
                    ...
                </otherwise>
            </switch>

            <!-- Return the response to the client -->
            <reply partnerLink="client"
                portType="aClientPort"
                operation="GetOffer"
                variable="providerResponse"/>
        </sequence>
    </process>
```

2.21. 0MQ

2.21.1. Sockets

2.21.1.1. Socket Creation Functions

Low Level Functions.

```
void *zmq_socket (void *context, int type);
int zmq_close (void *socket);
```

context	initialized library handle
type	role in communication pattern

High Level Functions.

```
// Generic socket creation functions.
zsock_t *zsock_new (int type);
void zsock_destroy (zsock_t **self_p);

// Type specific socket creation functions.
```

```
zsock_t *zsock_new_pub (const char *endpoint);
zsock_t *zsock_new_sub (const char *endpoint, const char *subscribe);
zsock_t *zsock_new_req (const char *endpoint);
zsock_t *zsock_new_rep (const char *endpoint);
```

2.21.1.2. Socket Connection Functions

Low Level Functions.

```
int zmq_bind (void *socket, const char *endpoint);
int zmq_connect (void *socket, const char *endpoint);
```

inproc	communication within process • address is an arbitrary string
ipc	communication over local pipe • address is a local pipe file name
vmci	communication over hypervisor transport • address is a virtual machine identifier or hypervisor • address includes port number with function similar to TCP or UDP
tcp	TCP socket opened on demand
pgm	IP multicast to destination
epgm	UDP multicast to destination

High Level Functions.

```
int zsock_bind (zsock_t *self, const char *format, ...);
int zsock_connect (zsock_t *self, const char *format, ...);
```

2.21.1.3. Socket Configuration Functions

Low Level Functions.

```
int zmq_setsockopt (void *socket, int option_name, const void *option_value, si
int zmq_getsockopt (void *socket, int option_name, void *option_value, size_t *
```

ZMQ_SUBSCRIBE	subscription filter
ZMQ_SNDHWM, ZMQ_RCVHWM	high water mark
ZMQ_SNDBUF, ZMQ_RCVBUF	system buffer size
ZMQ_RECONNECT_IVL	transport reconnect interval
ZMQ_RECOVERY_IVL	multicast absence tolerance
ZMQ_AFFINITY	transport thread affinity
ZMQ_RATE	multicast data rate

High Level Functions.

```
// Convert from high level to low level socket reference.
void *zsock_resolve (void *self);
```

2.21.1.4. Message Transport Functions

Low Level Functions.

```
int zmq_msg_send (zmq_msg_t *msg, void *socket, int flags);
int zmq_msg_recv (zmq_msg_t *msg, void *socket, int flags);
```

- messages are byte arrays

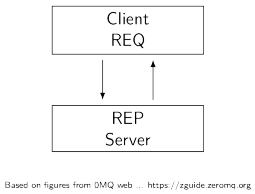
- message delivery is atomic
- multipart messages are supported

High Level Functions.

```
zmsg_t *zmsg_recv (void *source);
int zmsg_send (zmsg_t **self_p, void *dest);

// Multipart message functions.
size_t zmsg_size (zmsg_t *self);
int zmsg_prepend (zmsg_t *self, zframe_t **frame_p);
int zmsg_append (zmsg_t *self, zframe_t **frame_p);
zframe_t *zmsg_pop (zmsg_t *self);
zframe_t *zmsg_first (zmsg_t *self);
zframe_t *zmsg_next (zmsg_t *self);
```

2.21.2. Patterns

2.21.2.1. Synchronous Request Reply Pattern



Based on figures from 0MQ web ... https://zguide.zeromq.org

Connects multiple clients to multiple servers.

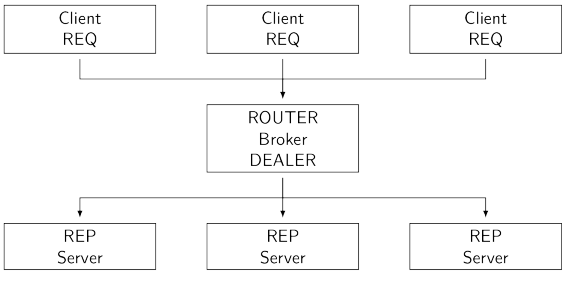
ZMQ_REQ.

- used by client to send requests and receive replies
- allows only alternating sequence of send and recv
- round robin when multiple servers connected
- blocks when no service available

ZMQ_REP.

- used by service to receive requests and send replies
- allows only alternating sequence of recv and send
- fair queueing among clients

2.21.2.2. Asynchronous Request Reply Pattern



Based on figures from 0MQ web ... https://zguide.zeromq.org

Connects multiple clients to multiple servers through an intermediary.

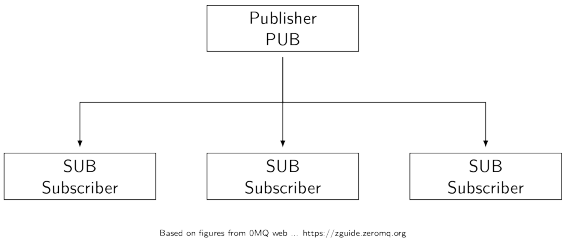
ZMQ_ROUTER.

- receives requests from clients with fair queueing
- prefixes requests with client identifier
- delivers replies to identified client

ZMQ_DEALER.

- receives replies from servers with fair queueing
- delivers requests to servers with round robin

2.21.2.3. Static Publish Subscribe Pattern



Connects multiple publishers to multiple subscribers.

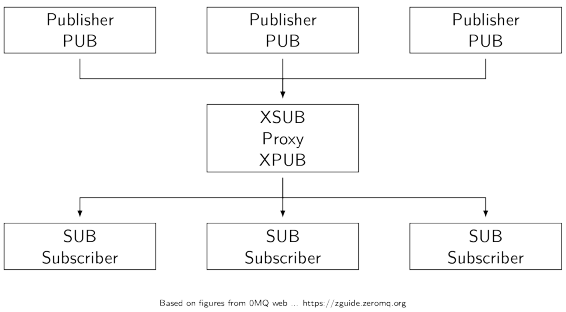
ZMQ_PUB.

- delivers messages to connected subscribers
- never blocks

ZMQ_SUB.

- receives messages from connected publishers
- fair queueing among publishers

2.21.2.4. Dynamic Publish Subscribe Pattern



Connects multiple publishers to multiple subscribers through an intermediary.

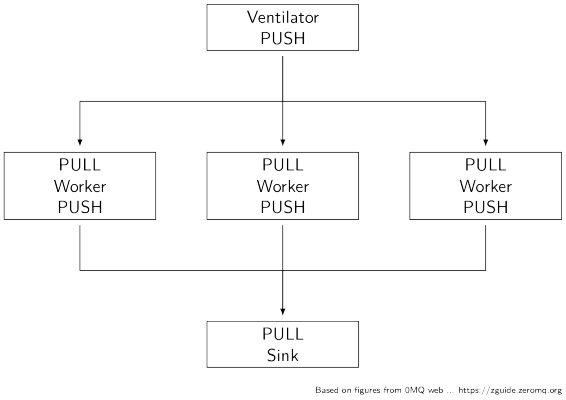
ZMQ_XSUB.

- delivers subscription requests to connected publishers
- receives messages from connected publishers
- fair queueing among publishers

ZMQ_XPUB.

- receives subscription requests from connected subscribers
- delivers messages to connected subscribers
- fair queueing among subscribers
- never blocks

2.21.2.5. Pipeline Pattern



Connects task generators to task processors.

ZMQ_PUSH.

- delivers messages to connected task processors
- round robin among processors

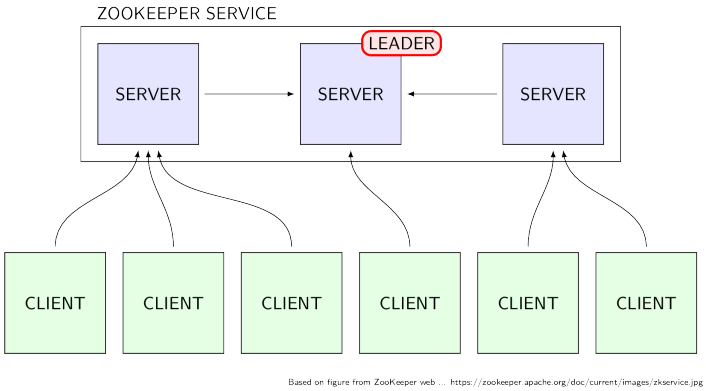
ZMQ_PULL.

- receives messages from connected task generators
- fair queueing among generators

2.22. Apache ZooKeeper

2.22.1. Architecture

2.22.1.1. ZooKeeper Architecture



Servers. Replicated server cluster

- Each server stores complete state in memory
- Updates are also stored in persistent log
- Persistent snapshot done when updates accumulate

Atomic communication protocol

- All updates pass through leader server
- Leader collects majority quorum for each update
- Leader election triggered in case of cluster failure

Clients.

- Provided with a list of servers to use
- Connected to a single server at a time
- Connection failure handled by switching to another server

2.22.2. Interface

2.22.2.1. ZooKeeper Data Model

Data.

- Tree of named nodes navigated by string paths
- Support for unique node naming
- Node data is array of bytes
- Updates increment version

2.22.2.2. ZooKeeper Data Objects

```
module org.apache.zookeeper.data {
    ...
    class Stat {
        long czxid;            // ZXID of transaction that created this node
        long mzxid;            // ZXID of transaction that last modified this
        long pzxid;            // ZXID of transaction that last modifined node
        long ctime;            // Node creation time
        long mtime;            // Node last modification time
        int version;           // Node version
        int aversion;          // Node ACL version
        int cversion;          // Node child version
        int dataLength;        // Node data length
        int numChildren;       // Node child count
        long ephemeralOwner;   // Owner identifier for ephemeral nodes
    }
    ...
}
```

2.22.2.3. ZooKeeper Blocking Interface

```
public class ZooKeeper {
    public ZooKeeper (String connectString, int sessionTimeout, Watcher watcher
    public ZooKeeper (String connectString, int sessionTimeout, Watcher watcher,
    ...

    public String create (String path, byte data [], List<ACL> acl, CreateMode
    public void delete (String path, int version) { ... }
```

```
    public Stat exists (String path, boolean watch) { ... }
    public Stat exists (String path, Watcher watcher) { ... }

    public byte [] getData (String path, boolean watch, Stat stat) { ... }
    public byte [] getData (String path, Watcher watcher, Stat stat) { ... }

    public Stat setData (String path, byte data [], int version) { ... }

    public List<String> getChildren (String path, boolean watch) { ... }
    public List<String> getChildren (String path, boolean watch, Stat stat) { .
    public List<String> getChildren (String path, Watcher watcher) { ... }
    public List<String> getChildren (String path, Watcher watcher, Stat stat) {

    // Make sure the server is current with the leader.
    public void sync (String path, VoidCallback cb, Object ctx) { ... }

    public synchronized void close () { ... }
}
```

2.22.2.4. ZooKeeper Non Blocking Interface

```
public class ZooKeeper {
    ...

    public void create (
        String path, byte data [],
        List<ACL> acl, CreateMode createMode,
        StringCallback cb, Object ctx) { ... }

    public void delete(String path, int version, VoidCallback cb, Object ctx) {

    public void exists (String path, boolean watch, StatCallback cb, Object ctx
    public void exists (String path, Watcher watcher, StatCallback cb, Object c

    public void getData (String path, boolean watch, DataCallback cb, Object ct
    public void getData (String path, Watcher watcher, DataCallback cb, Object

    public void setData (String path, byte data [], int version, StatCallback c

    public void getChildren (String path, boolean watch, ChildrenCallback cb, O
    public void getChildren (String path, boolean watch, Children2Callback cb,
    public void getChildren (String path, Watcher watcher, ChildrenCallback cb,
    public void getChildren (String path, Watcher watcher, Children2Callback cb

    ...
}

public interface StatCallback extends AsyncCallback {
    public void processResult (int rc, String path, Object ctx, Stat stat);
}

public interface DataCallback extends AsyncCallback {
    public void processResult (int rc, String path, Object ctx, byte data [], S
}

public interface ChildrenCallback extends AsyncCallback {
```

```
        public void processResult (int rc, String path, Object ctx, List<String> ch
    }

    public interface Children2Callback extends AsyncCallback {
        public void processResult (int rc, String path, Object ctx, List<String> ch
    }

    ...
```

2.22.2.5. ZooKeeper Multiple Operations Interface

```
    public class ZooKeeper {
        ...

        // Execute multiple operations atomically.
        public List<OpResult> multi (Iterable<Op> ops) { ... }
        public void multi (Iterable<Op> ops, MultiCallback cb, Object ctx) { ... }

        ...
    }

    public abstract class Op {
        private int type;
        private String path;

        private Op (int type, String path) {
            this.type = type;
            this.path = path;
        }

        public static Op create (String path, byte [] data, List<ACL> acl, int flags
            return new Create (path, data, acl, flags);
        }

        public static class Create extends Op {
            private byte [] data;
            private List<ACL> acl;
            private int flags;

            private Create (String path, byte [] data, List<ACL> acl, int flags) {
                super (ZooDefs.OpCode.create, path);
                this.data = data;
                this.acl = acl;
                this.flags = flags;
            }

            ...
        }

        ...
    }


    public abstract class OpResult {
        private int type;

        private OpResult (int type) {
```

```
            this.type = type;
        }

        public static class CreateResult extends OpResult {
            private String path;

            public CreateResult (String path) {
                super (ZooDefs.OpCode.create);
                this.path = path;
            }

            ...
        }

        ...
    }
```

2.22.2.6. ZooKeeper Watcher Interface

```
    public class ZooKeeper {
        ...

        // Manage watches with explicit mode.
        void addWatch (String basePath, AddWatchMode mode);
        void removeWatches (String path, Watcher watcher, Watcher.WatcherType watch

        ...
    }


    public enum AddWatchMode {
        PERSISTENT (0),
        PERSISTENT_RECURSIVE (1);
    }


    public interface Watcher {

        abstract public void process (WatchedEvent event);

        public interface Event {
            public enum EventType {
                None (-1),
                NodeCreated (1),
                NodeDeleted (2),
                NodeDataChanged (3),
                NodeChildrenChanged (4);

                ...
            }
        }
    }


    public class WatchedEvent {
        ...
```

```
    public KeeperState getState () { ... }
    public EventType getType () { ... }
    public String getPath () { ... }
}
```

- One shot watches are removed after every event
- Persistent watches stay until removed explicitly
- Recursive watches also report events on children

2.22.3. Recipes

2.22.3.1. Curator Recipes

Agreement.

GroupMember	group membership tracking
LeaderLatch	leader election with polling interface
LeaderSelector	leader election with callback interface

Synchronization.

DistributedBarrier	barrier with explicit state setting calls
DistributedDoubleBarrier	barrier with node count condition
InterProcessMutex	recursive lock
InterProcessSemaphoreMutex	non recursive lock
InterProcessReadWriteLock	recursive read write lock
InterProcessSemaphore	semaphore
InterProcessMultilock	wrapper for acquiring multiple locks atomically

Communication.

SimpleDistributedQueue	backwards compatible queue
DistributedQueue	ordered queue with optional item identities
DistributedDelayQueue	queue with delayed delivery
DistributedPriorityQueue	queue with priorities
SharedCount	shared integer counter
DistributedAtomicLong	shared long integer counter

Resiliency.

CuratorCache	generic local path cache
PersistentNode	connection loss resistant node interface
PersistentTTLNode	connection loss resistant node interface with keepalive
PersistentWatcher	connection loss resistant watch interface