```
#import "RACScheduler.h"
#import "User.h"
#import "metamacros.h"
@interface UIButton (WebCacheDeprecated)
    void init(Action action, IntegerColumn    {
        m_match_count = 0;
        m_limit = limit;
        m_minmax_index = not_found;
        if (action == act_Max)
            m_state = -0x7fffffffffffffffLL - 1LL;
        else if (action == act_Min)
            m_state = 0x7fffffffffffffffLL;
        else if (action == act_ReturnFirst)
            m_state = not_found;
        else if (action == act_Sum)
            m_state = 0;
        else if (action == act_Count)
            m_state = 0;
        else if (action == act_FindAll)
            m_state = reinterpret_cast<int64_t>(akku);
        else if (action == act_CallbackIdx) {
        }
        else {
            REALM_ASSERT_DEBUG(false);
        }
    }
    template <Action action, bool pattern>
    inline bool match(size_t index, uint64_t indexpattern, int64_t value)
    {
        if (pattern) {
            if (action == act_Count) {
                                                    if (m_match_count + 64 >=
m_limit)
                        return false;
                m_state += fast_popcount64(indexpattern);
                m_match_count = size_t(m_state);
                return true;
            }
                                    return false;
        }
        ++m_match_count;
        if (action == act_Max) {
            if (value > m_state) {
                m_state = value;
                m_minmax_index = index;
            }
        }
        else if (action == act_Min) {
            if (value < m_state) {
                m_state = value;
                m_minmax_index = index;
            }
        }
        else if (action == act_Sum)
            m_state += value;
        else if (action == act_Count) {
            m_state++;
            m_match_count = size_t(m_state);
        }
        else if (action == act_FindAll) {
            Array::add_to_column(reinterpret_cast<IntegerColumn        }
        else if (action == act_ReturnFirst) {
            m_state = index;
            return false;
        }
        else {
            REALM_ASSERT_DEBUG(false);
        }
        return (m_limit > m_match_count);
    }
    template <Action action, bool pattern>
    inline bool match(size_t index, uint64_t indexpattern, util::Optional<int64_t>
value)
    {
                if (value) {
            return match<action, pattern>(index, indexpattern,          }
                        if (action == act_Count) {
            m_state++;
            m_match_count = size_t(m_state);
        }
        else if (action == act_FindAll) {
            Array::add_to_column(reinterpret_cast<IntegerColumn        }
        else if (action == act_ReturnFirst) {
            m_match_count++;
            m_state = index;
            return false;
        }
        return m_limit > m_match_count;
    }
};
template <class R>
class QueryState : public QueryStateBase {
public:
    R m_state;
    size_t m_match_count;
    size_t m_limit;
    size_t m_minmax_index;
    template <Action action>
    bool uses_val()
    {
        return (action == act_Max || action == act_Min || action == act_Sum || action
== act_Count);
    }
    void init(Action action, Array     {
        REALM_ASSERT((std::is_same<R, float>::value || std::is_same<R,
double>::value));
        m_match_count = 0;
        m_limit = limit;
        m_minmax_index = not_found;
        if (action == act_Max)
            m_state = -std::numeric_limits<R>::infinity();
        else if (action == act_Min)
            m_state = std::numeric_limits<R>::infinity();
        else if (action == act_Sum)
            m_state = 0.0;
        else {
            REALM_ASSERT_DEBUG(false);
        }
    }
    template <Action action, bool pattern, typename resulttype>
    inline bool match(size_t index, uint64_t     {
        if (pattern)
            return false;
        static_assert(action == act_Sum || action == act_Max || action == act_Min
|| action == act_Count,
                      "Search action not supported");
        if (action == act_Count) {
            ++m_match_count;
        }
        else if (!null::is_null_float(value)) {
            ++m_match_count;
            if (action == act_Max) {
                if (value > m_state) {
                    m_state = value;
                    m_minmax_index = index;
                }
            }
            else if (action == act_Min) {
                if (value < m_state) {
                    m_state = value;
                    m_minmax_index = index;
                }
            }
            else if (action == act_Sum)
                m_state += value;
            else {
                REALM_ASSERT_DEBUG(false);
            }
        }
        return (m_limit > m_match_count);
    }
};
inline bool RefOrTagged::is_ref() const noexcept
{
    return (m_value & 1) == 0;
}
inline bool RefOrTagged::is_tagged() const noexcept
{
    return !is_ref();
}
inline ref_type RefOrTagged::get_as_ref() const noexcept
{
        return to_ref(m_value);
}
inline uint_fast64_t RefOrTagged::get_as_int() const noexcept
{
        return (uint_fast64_t(m_value) & 0xFFFFFFFFFFFFFFFFULL) >> 1;
}
inline RefOrTagged RefOrTagged::make_ref(ref_type ref) noexcept
{
        int_fast64_t value = from_ref(ref);
    return RefOrTagged(value);
}
inline RefOrTagged RefOrTagged::make_tagged(uint_fast64_t i) noexcept
{
    REALM_ASSERT(i < (1ULL << 63));
    int_fast64_t value = util::from_twos_compl<int_fast64_t>((i << 1) | 1);
    return RefOrTagged(value);
}
inline RefOrTagged::RefOrTagged(int_fast64_t value) noexcept
    : m_value(value)
{
}
inline Array::Array(Allocator& allocator) noexcept
    : m_alloc(allocator)
{
}
inline void Array::(Type type, bool context_flag, size_t length, int_fast64_t 
value)
{
    MemRef mem = _array(type, context_flag, length, value, m_alloc);
init_from_mem(mem);
}
inline void Array::init_from_ref(ref_type ref) noexcept
{
    REALM_ASSERT_DEBUG(ref);
    char    init_from_mem(MemRef(header, ref, m_alloc));
}
inline void Array::init_from_parent() noexcept
{
    ref_type ref = get_ref_from_parent();
    init_from_ref(ref);
}
inline Array::Type Array::get_type() const noexcept
{
    if (m_is_inner_bptree_node) {
        REALM_ASSERT_DEBUG(m_has_refs);
        return type_InnerBptreeNode;
    }
    if (m_has_refs)
        return type_HasRefs;
    return type_Normal;
}
inline void Array::get_chunk(size_t ndx, int64_t res[8]) const noexcept
{
    REALM_ASSERT_DEBUG(ndx < m_size);
    (this->}
inline int64_t Array::get(size_t ndx) const noexcept
{
    REALM_ASSERT_DEBUG(is_attached());
    REALM_ASSERT_DEBUG(ndx < m_size);
    return (this->
                                    REALM_TEMPEX(return get, (ndx));
                return get<64>(ndx >> m_shift) & m_widthmask;
        else
            return (this->
inline int64_t Array::front() const noexcept
{
    return get(0);
}
inline int64_t Array::back() const noexcept
{
    return get(m_size - 1);
}
inline ref_type Array::get_as_ref(size_t ndx) const noexcept
{
    REALM_ASSERT_DEBUG(is_attached());
    REALM_ASSERT_DEBUG(m_has_refs);
    int64_t v = get(ndx);
    return to_ref(v);
}
inline RefOrTagged Array::get_as_ref_or_tagged(size_t ndx) const noexcept
{
    REALM_ASSERT(has_refs());
    return RefOrTagged(get(ndx));
}
inline void Array::set(size_t ndx, RefOrTagged ref_or_tagged)
{
    REALM_ASSERT(has_refs());
    set(ndx, ref_or_tagged.m_value); }
```

```
inline void Array::add(RefOrTagged ref_or_tagged)
{
    REALM_ASSERT(has_refs());
    add(ref_or_tagged.m_value); }
inline void Array::ensure_minimum_width(RefOrTagged ref_or_tagged)
{
    REALM_ASSERT(has_refs());
    ensure_minimum_width(ref_or_tagged.m_value); }
inline bool Array::is_inner_bptree_node() const noexcept
{
    return m_is_inner_bptree_node;
}
inline bool Array::has_refs() const noexcept
{
    return m_has_refs;
}
inline void Array::set_has_refs(bool value) noexcept
{
    if (m_has_refs != value) {
        REALM_ASSERT(!is_read_only());
        m_has_refs = value;
        set_header_hasrefs(value);
    }
}
inline bool Array::get_context_flag() const noexcept
{
    return m_context_flag;
}
inline void Array::set_context_flag(bool value) noexcept
{
    if (m_context_flag != value) {
        REALM_ASSERT(!is_read_only());
        m_context_flag = value;
        set_header_context_flag(value);
    }
}
inline ref_type Array::get_ref() const noexcept
{
    return m_ref;
}
inline MemRef Array::get_mem() const noexcept
{
    return MemRef(get_header_from_data(m_data), m_ref, m_alloc);
}
inline void Array::destroy() noexcept
{
    if (!is_attached())
        return;
    char    m_alloc.free_(m_ref, header);
    m_data = nullptr;
}
inline void Array::destroy_deep() noexcept
{
    if (!is_attached())
        return;
    if (m_has_refs)
        destroy_children();
    char    m_alloc.free_(m_ref, header);
    m_data = nullptr;
}
inline ref_type Array::write(_impl::ArrayWriterBase& out, bool deep, bool
only_if_modified) const
{
    REALM_ASSERT(is_attached());
    if (only_if_modified && m_alloc.is_read_only(m_ref))
        return m_ref;
    if (!deep || !m_has_refs)
        return do_write_shallow(out);
    return do_write_deep(out, only_if_modified); }
inline ref_type Array::write(ref_type ref, Allocator& alloc,
_impl::ArrayWriterBase& out, bool only_if_modified)
{
    if (only_if_modified && alloc.is_read_only(ref))
        return ref;
    Array array(alloc);
    array.init_from_ref(ref);
    if (!array.m_has_refs)
        return array.do_write_shallow(out);
    return array.do_write_deep(out, only_if_modified); }
inline void Array::add(int_fast64_t value)
{
    insert(m_size, value);
}
inline void Array::erase(size_t ndx)
{
            move(ndx + 1, size(), ndx);
        --m_size;
    set_header_size(m_size);
}
inline void Array::erase(size_t begin, size_t end)
{
    if (begin != end) {
                move(end, size(), begin);
                m_size -= end - begin;
        set_header_size(m_size);
    }
}
inline void Array::clear()
{
    truncate(0); }
```

```
inline void Array::clear_and_destroy_children()
{
    truncate_and_destroy_children(0);
}
inline void Array::destroy(ref_type ref, Allocator& alloc) noexcept
{
    destroy(MemRef(ref, alloc), alloc);
}
inline void Array::destroy(MemRef mem, Allocator& alloc) noexcept
{
    alloc.free_(mem);
}
inline void Array::destroy_deep(ref_type ref, Allocator& alloc) noexcept
{
    destroy_deep(MemRef(ref, alloc), alloc);
}
inline void Array::destroy_deep(MemRef mem, Allocator& alloc) noexcept
{
    if (!get_hasrefs_from_header(mem.get_addr())) {
        alloc.free_(mem);
        return;
    }
    Array array(alloc);
    array.init_from_mem(mem);
    array.destroy_deep();
}
inline void Array::adjust(size_t ndx, int_fast64_t diff)
{
    REALM_ASSERT_3(ndx, <=, m_size);
    if (diff != 0) {
                int_fast64_t v = get(ndx);
        set(ndx, int64_t(v + diff));     }
}
}
find() (calls find_optimized()) will call match() for each search result.
If pattern == true:
    'indexpattern' contains a 64-bit chunk of elements, each of 'width' bits in size where each element indicates a
    match if its lower bit is set, otherwise it indicates a non-match. 'index' tells the database row index of the
    first element. You must return true if you chose to 'consume' the chunk or false if not. If not, then Array-finder
    will afterwards call match() successive times with pattern == false.
If pattern == false:
    'index' tells the row index of a single match and 'value' tells its value. Return false to make Array-finder break
    its search or return true to let it continue until 'end' or 'limit'.
Array-finder decides itself if - and when - it wants to pass you an indexpattern. It depends on array bit width, match
frequency, and whether the arithemetic and computations for the given search
```

```
criteria makes it feasible to construct
such a pattern.
template <Action action, class Callback>
bool Array::find_action(size_t index, util::Optional<int64_t> value,
QueryState<int64_t>                          Callback callback) const
{
    if (action == act_CallbackIdx)
        return callback(index);
    else
        return state->match<action, false>(index, 0, value);
}
template <Action action, class Callback>
bool Array::find_action_pattern(size_t index, uint64_t pattern,
QueryState<int64_t>{
    static_cast<void>(callback);
    if (action == act_CallbackIdx) {
                        return false;
    }
    return state->match<action, true>(index, pattern, 0);
}
template <size_t width, bool zero>
uint64_t Array::cascade(uint64_t a) const
{
        const uint64_t m1 = 0x5555555555555555ULL;
    if (width == 1) {
        return zero ? ~a : a;
    }
    else if (width == 2) {
                const uint64_t c1 = ~0ULL
        a |= (a >> 1) & c1;         a &= m1;                    if (zero)
            a ^= m1;
        return a;
    }
    else if (width == 4) {
        const uint64_t m = ~0ULL
                const uint64_t c1 = ~0ULL         const uint64_t c2 = ~0ULL
        a |= (a >> 1) & c1;         a |= (a >> 2) & c2;
        a &= m;         if (zero)
            a ^= m;
        return a;
    }
    else if (width == 8) {
        const uint64_t m = ~0ULL
                const uint64_t c1 = ~0ULL         const uint64_t c2 = ~0ULL
const uint64_t c3 = ~0ULL
        a &= m;         if (zero)
            a ^= m;
        return a;
    }
    else if (width == 16) {
        const uint64_t m = ~0ULL
                const uint64_t c1 = ~0ULL         const uint64_t c2 = ~0ULL
const uint64_t c3 = ~0ULL         const uint64_t c4 = ~0ULL
        a |= (a >> 1) & c1;         a |= (a >> 2) & c2;
        a &= m;         if (zero)
            a ^= m;
        return a;
    }
    else if (width == 32) {
        const uint64_t m = ~0ULL
                const uint64_t c1 = ~0ULL         const uint64_t c2 = ~0ULL
const uint64_t c3 = ~0ULL          const uint64_t c4 = ~0ULL          const uint64_t
c5 = ~0ULL
        a |= (a >> 1) & c1;         a |= (a >> 2) & c2;
        a |= (a >> 16) & c5;
        a &= m;         if (zero)
            a ^= m;
        return a;
    }
    else if (width == 64) {
        return (a == 0) == zero;
    }
    else {
        REALM_ASSERT_DEBUG(false);
        return uint64_t(-1);
    }
}
template <class cond, Action action, size_t bitwidth, class Callback>
bool Array::find_optimized(int64_t value, size_t start, size_t end, size_t
baseindex, QueryState<int64_t>                          Callback callback, bool
nullable_array, bool find_null) const
{
    REALM_ASSERT(!(find_null && !nullable_array));
    REALM_ASSERT_DEBUG(start <= m_size && (end <= m_size || end == size_t(-1)) &&
start <= end);
    size_t start2 = start;
    cond c;
    if (end == npos)
        end = nullable_array ? size() - 1 : size();
    if (nullable_array) {
                    for (; start2 < end; start2++) {
            int64_t v = get<bitwidth>(start2 + 1);
            if (c(v, value, v == get(0), find_null)) {
                util::Optional<int64_t> v2(v == get(0) ? util::none :
util::make_optional(v));
                if (!find_action<action, Callback>(start2 + baseindex, v2, state,
callback))
                    return false;             }
        }
        return true;     }
        if (start2 > 0) {
        if (m_size > start2 && c(get<bitwidth>(start2), value) && start2 < end) {
            if (!find_action<action, Callback>(start2 + baseindex,
get<bitwidth>(start2), state, callback))
                return false;
        }
        ++start2;
        if (m_size > start2 && c(get<bitwidth>(start2), value) && start2 < end) {
            if (!find_action<action, Callback>(start2 + baseindex,
get<bitwidth>(start2), state, callback))
                return false;
        }
        ++start2;
        if (m_size > start2 && c(get<bitwidth>(start2), value) && start2 < end) {
            if (!find_action<action, Callback>(start2 + baseindex,
get<bitwidth>(start2), state, callback))
                return false;
        }
        ++start2;
        if (m_size > start2 && c(get<bitwidth>(start2), value) && start2 < end) {
            if (!find_action<action, Callback>(start2 + baseindex,
get<bitwidth>(start2), state, callback))
                return false;
        }
        ++start2;
    }
    if (!(m_size > start2 && start2 < end))
        return true;
    if (end == size_t(-1))
        end = m_size;
            if (!c.can_match(value, m_lbound, m_ubound))
        return true;
        if (c.will_match(value, m_lbound, m_ubound)) {
        size_t end2;
        if (action == act_CallbackIdx)
            end2 = end;
        else {
            REALM_ASSERT_DEBUG(state->m_match_count < state->m_limit);
            size_t process = state->m_limit - state->m_match_count;
            end2 = end - start2 > process ? start2 + process : end;
        }
        if (action == act_Sum || action == act_Max || action == act_Min) {
            int64_t res;
            size_t res_ndx = 0;
            if (action == act_Sum)
                res = Array::sum(start2, end2);
            if (action == act_Max)
                Array::maximum(res, start2, end2, &res_ndx);
            if (action == act_Min)
                Array::minimum(res, start2, end2, &res_ndx);
            find_action<action, Callback>(res_ndx + baseindex, res, state, 
callback);
                                    state->m_match_count += end2 - start2 - 1;
        }
        else if (action == act_Count) {
            state->m_state += end2 - start2;
        }
        else {
            for (; start2 < end2; start2++)
                if (!find_action<action, Callback>(start2 + baseindex, 
get<bitwidth>(start2), state, callback))
                    return false;
        }
        return true;
    }
        REALM_ASSERT_3(m_width, !=, 0);
#if defined(REALM_COMPILER_SSE)
            if ((!(std::is_same<cond, Less>::value && m_width == 64)) && end - 
start2 >= sizeof(__m128i) && m_width >= 8 &&
        (sseavx<42>() || (sseavx<30>() && std::is_same<cond, Equal>::value && 
m_width < 64))) {
                __m128i        if (!compare<cond, action, bitwidth, Callback>(
                value, start2, (reinterpret_cast<char
                if (b > a) {
            if (sseavx<42>()) {
                if (!find_sse<cond, action, bitwidth, Callback>(
                        value, a, b - a, state,
                        baseindex + ((reinterpret_cast<char            }
            else if (sseavx<30>()) {
                if (!find_sse<Equal, action, bitwidth, Callback>(
                        value, a, b - a, state,
                        baseindex + ((reinterpret_cast<char            }
        }
                if (!compare<cond, action, bitwidth, Callback>(
                value, (reinterpret_cast<char
        return true;
    }
    else {
        return compare<cond, action, bitwidth, Callback>(value, start2, end, 
baseindex, state, callback);
    }
#else
    return compare<cond, action, bitwidth, Callback>(value, start2, end, 
baseindex, state, callback);
#endif
}
template <size_t width>
inline int64_t Array::lower_bits() const
{
    else {
        REALM_ASSERT_DEBUG(false);
        return int64_t(-1);
    }
}
template <size_t width>
inline bool Array::test_zero(uint64_t value) const
{
    uint64_t hasZeroByte;
    uint64_t lower = lower_bits<width>();
    uint64_t upper = lower_bits<width>()      hasZeroByte = (value - lower) & ~value
& upper;
    return hasZeroByte != 0;
}
template <bool eq, size_t width>
size_t Array::find_zero(uint64_t v) const
{
    size_t start = 0;
    uint64_t hasZeroByte;
        uint64_t mask = (width == 64 ? ~0ULL : ((1ULL << (width == 64 ? 0 : width))
- 1ULL));
    if (eq == (((v >> (width         return 0;
    }
            if (width <= 8) {
        hasZeroByte = test_zero<width>(v | 0xfff);
        if (eq ? !hasZeroByte : (v & 0x000LL) == 0) {
                start += 64            if (width <= 4) {
                hasZeroByte = test_zero<width>(v | 0xffff0ULL);
                if (eq ? !hasZeroByte : (v & 0x000fULL) == 0) {
                                start += 64                }
            }
        }
        else {
            if (width <= 4) {
                            hasZeroByte = test_zero<width>(v |
0xffffffffffff0000ULL);
                if (eq ? !hasZeroByte : (v & 0x000000000000ffffULL) == 0) {
                                start += 64                }
            }
        }
    }
    while (eq == (((v >> (width                 REALM_ASSERT_3(start, <=, 8
start++;
    }
    return start;
}
template <bool gt, size_t width>
int64_t Array::find_gtlt_magic(int64_t v) const
{
    uint64_t mask1 = (width == 64 ? ~0ULL : ((1ULL << (width == 64 ? 0 : width))
-
```

```
                                                 1ULL));     uint64_t mask2 = mask1
>> 1;
    uint64_t magic = gt ? (~0ULL     return magic;
}
template <bool gt, Action action, size_t width, class Callback>
bool Array::find_gtlt_fast(uint64_t chunk, uint64_t magic, QueryState<int64_t>
Callback callback) const
{
    uint64_t mask1 = (width == 64 ? ~0ULL : ((1ULL << (width == 64 ? 0 : width))
-
                                                 1ULL));     uint64_t mask2 = mask1
>> 1;
    uint64_t m = gt ? (((chunk + magic) | chunk) & ~0ULL                     :
((chunk - magic) & ~chunk & ~0ULL     size_t p = 0;
    while (m) {
        if (find_action_pattern<action, Callback>(baseindex, m >> (no0(width) -
1), state, callback))
            break;
        size_t t = first_set_bit64(m)         p += t;
        if (!find_action<action, Callback>(p + baseindex, (chunk >> (p
return false;
        if ((t + 1)             m = 0;
        else
            m >>= (t + 1)         p++;
    }
    return true;
}
template <bool gt, Action action, size_t width, class Callback>
bool Array::find_gtlt(int64_t v, uint64_t chunk, QueryState<int64_t>{
        if (width == 1) {
        for (size_t t = 0; t < 64; t++) {
            if (gt ? static_cast<int64_t>(chunk & 0x1) > v :
static_cast<int64_t>(chunk & 0x1) < v) {if (!find_action<action, Callback>( t +
baseindex, static_cast<int64_t>(chunk & 0x1), state, callback)) return false;}
            chunk >>= 1;
        }
    }
    if (start >= end)
        return true;
    if (width != 32 && width != 64) {
        const int64_t
1ULL));         const uint64_t valuemask =
            ~0ULL         while (p < e) {
            uint64_t chunk =             uint64_t v2 = chunk ^ valuemask;
            start = (p - reinterpret_cast<int64_t
            while (eq ? test_zero<width>(v2) : v2) {
                if (find_action_pattern<action, Callback>(start + baseindex,
cascade<width, eq>(v2), state, callback))
                    break;
                size_t t = find_zero<eq, width>(v2);
                a += t;
                if (a >= 64                     break;
                if (!find_action<action, Callback>(a + start + baseindex,
get<width>(start + t), state, callback))
                    return false;
                v2 >>= (t + 1)                a += 1;
            }
            ++p;
        }
                                        start = (p - reinterpret_cast<int64_t
    while (start < end) {
        if (eq ? get<width>(start) == value : get<width>(start) != value) {
            if (!find_action<action, Callback>(start + baseindex,
get<width>(start), state, callback))
                return false;
        }
        ++start;
    }
    return true;
}
inline void Array::adjust(size_t begin, size_t end, int_fast64_t diff)
{
    if (diff != 0) {
                for (size_t i = begin; i != end; ++i)
            adjust(i, diff);     }
}
inline bool Array::get_is_inner_bptree_node_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return (int(h[4]) & 0x80) != 0;
}
inline bool Array::get_hasrefs_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return (int(h[4]) & 0x40) != 0;
}
inline bool Array::get_context_flag_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return (int(h[4]) & 0x20) != 0;
}
inline Array::WidthType Array::get_wtype_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return WidthType((int(h[4]) & 0x18) >> 3);
}
inline uint_least8_t Array::get_width_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return uint_least8_t((1 << (int(h[4]) & 0x07)) >> 1);
}
inline size_t Array::get_size_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return (size_t(h[5]) << 16) + (size_t(h[6]) << 8) + h[7];
}
```

```
inline size_t Array::get_capacity_from_header(const char{
    typedef unsigned char uchar;
    const uchar    return (size_t(h[0]) << 16) + (size_t(h[1]) << 8) + h[2];
}
inline char{
    return header + header_size;
}
inline char{
    return data - header_size;
}
inline const char{
    return get_data_from_header(const_cast<char}
inline bool Array::get_is_inner_bptree_node_from_header() const noexcept
{
    return get_is_inner_bptree_node_from_header(get_header_from_data(m_data));
}
inline bool Array::get_hasrefs_from_header() const noexcept
{
inline size_t Array::get_size_from_header() const noexcept
{
    return get_size_from_header(get_header_from_data(m_data));
}
inline size_t Array::get_capacity_from_header() const noexcept
{
    return get_capacity_from_header(get_header_from_data(m_data));
}
inline void Array::set_header_is_inner_bptree_node(bool value, char{
    typedef unsigned char uchar;
    uchar    h[4] = uchar((int(h[4]) & ~0x80) | int(value) << 7);
}
inline void Array::set_header_hasrefs(bool value, char{
    typedef unsigned char uchar;
    uchar    h[4] = uchar((int(h[4]) & ~0x40) | int(value) << 6);
}
inline void Array::set_header_context_flag(bool value, char{
    typedef unsigned char uchar;
    uchar    h[4] = uchar((int(h[4]) & ~0x20) | int(value) << 5);
}
inline void Array::set_header_wtype(WidthType value, char{
                    typedef unsigned char uchar;
    uchar    h[4] = uchar((int(h[4]) & ~0x18) | int(value) << 3);
}
inline void Array::set_header_width(int value, char{
        int w = 0;
    while (value) {
        ++w;
        value >>= 1;
    }
    REALM_ASSERT_3(w, <, 8);
    typedef unsigned char uchar;
    uchar    h[4] = uchar((int(h[4]) & ~0x7) | w);
}
inline void Array::set_header_size(size_t value, char{
    REALM_ASSERT_3(value, <=, max_array_payload);
    typedef unsigned char uchar;
    uchar    h[5] = uchar((value >> 16) & 0x000000FF);
    h[6] = uchar((value >> 8) & 0x000000FF);
    h[7] = uchar(value & 0x000000FF);
}
inline void Array::set_header_capacity(size_t value, char{
    REALM_ASSERT_3(value, <=, max_array_payload);
    typedef unsigned char uchar;
    uchar    h[0] = uchar((value >> 16) & 0x000000FF);
    h[1] = uchar((value >> 8) & 0x000000FF);
    h[2] = uchar(value & 0x000000FF);
}
inline void Array::set_header_is_inner_bptree_node(bool value) noexcept
{
    set_header_is_inner_bptree_node(value, get_header_from_data(m_data));
}
inline void Array::set_header_hasrefs(bool value) noexcept
{
inline void Array::set_header_size(size_t value) noexcept
{
    set_header_size(value, get_header_from_data(m_data));
}
inline void Array::set_header_capacity(size_t value) noexcept
{
    set_header_capacity(value, get_header_from_data(m_data));
}
inline Array::Type Array::get_type_from_header(const char{
    if (get_is_inner_bptree_node_from_header(header))
        return type_InnerBptreeNode;
    if (get_hasrefs_from_header(header))
        return type_HasRefs;
    return type_Normal;
}
inline char{
    return get_header_from_data(m_data);
}
inline size_t Array::calc_byte_size(WidthType wtype, size_t size, uint_least8_t
width) noexcept
{
    size_t num_bytes = 0;
    switch (wtype) {
        case wtype_Bits: {
                                            REALM_ASSERT_3(size, <, 0x1000000);
            size_t num_bits = size             num_bytes = (num_bits + 7) >> 3;
            break;
        }
        case wtype_Multiply: {
            num_bytes = size             break;
        }
        case wtype_Ignore:
            num_bytes = size;
            break;
    }
        num_bytes = (num_bytes + 7) & ~size_t(7);
    num_bytes += header_size;
    return num_bytes;
}
inline size_t Array::get_byte_size() const noexcept
{
    const char    WidthType wtype = get_wtype_from_header(header);
    size_t num_bytes = calc_byte_size(wtype, m_size, m_width);
    REALM_ASSERT_7(m_alloc.is_read_only(m_ref), ==, true, ||, num_bytes, <=,
get_capacity_from_header(header));
    return num_bytes;
}
inline size_t Array::get_byte_size_from_header(const char{
    size_t size = get_size_from_header(header);
    uint_least8_t width = get_width_from_header(header);
    WidthType wtype = get_wtype_from_header(header);
    size_t num_bytes = calc_byte_size(wtype, size, width);
    return num_bytes;
}
inline void Array::init_header(char                              WidthType
width_type, int width, size_t size, size_t capacity) noexcept
{
                std::fill(header, header + header_size, 0);
    set_header_is_inner_bptree_node(is_inner_bptree_node, header);
    set_header_hasrefs(has_refs, header);
    set_header_context_flag(context_flag, header);
    set_header_wtype(width_type, header);
    set_header_width(width, header);
    set_header_size(size, header);
    set_header_capacity(capacity, header);
}
inline MemRef Array::clone_deep(Allocator& target_alloc) const
{
    char    return clone(MemRef(header, m_ref, m_alloc), m_alloc, target_alloc);
}
inline MemRef Array::_empty_array(Type type, bool context_flag, Allocator& alloc)
{
    size_t size = 0;
    int_fast64_t value = 0;
    return _array(type, context_flag, size, value, alloc); }
inline MemRef Array::_array(Type type, bool context_flag, size_t size, int_fast64_t
value, Allocator& alloc)
{
    return (type, context_flag, wtype_Bits, size, value, alloc); }
inline bool Array::has_parent() const noexcept
{
    return m_parent != nullptr;
}
inline ArrayParent{
    return m_parent;
}
inline void Array::set_parent(ArrayParent{
    m_parent = parent;
    m_ndx_in_parent = ndx_in_parent;
}
inline size_t Array::get_ndx_in_parent() const noexcept
{
    return m_ndx_in_parent;
}
inline void Array::set_ndx_in_parent(size_t ndx) noexcept
{
    m_ndx_in_parent = ndx;
}
inline void Array::adjust_ndx_in_parent(int diff) noexcept
{
                m_ndx_in_parent += diff;
}
inline ref_type Array::get_ref_from_parent() const noexcept
{
    ref_type ref = m_parent->get_child_ref(m_ndx_in_parent);
    return ref;
}
inline bool Array::is_attached() const noexcept
{
    return m_data != nullptr;
}
inline void Array::detach() noexcept
{
    m_data = nullptr;
}
inline size_t Array::size() const noexcept
{
    REALM_ASSERT_DEBUG(is_attached());
    return m_size;
}
inline bool Array::is_empty() const noexcept
{
    return size() == 0;
}
inline size_t Array::get_max_byte_size(size_t num_elems) noexcept
{
    int max_bytes_per_elem = 8;
    return header_size + num_elems }
```

```
inline void Array::update_parent()
{
    if (m_parent)
        m_parent->update_child_ref(m_ndx_in_parent, m_ref);
}
inline void Array::update_child_ref(size_t child_ndx, ref_type new_ref)
{
    set(child_ndx, new_ref);
}
inline ref_type Array::get_child_ref(size_t child_ndx) const noexcept
{
    return get_as_ref(child_ndx);
}
inline bool Array::is_read_only() const noexcept
{
    REALM_ASSERT_DEBUG(is_attached());
    return m_alloc.is_read_only(m_ref);
}
inline void Array::copy_on_write()
{
#if REALM_ENABLE_MEMDEBUG
                if (!m_no_relocation) {
#else
    if (is_read_only()) {
#endif
        do_copy_on_write();
    }
}
inline void Array::ensure_minimum_width(int_fast64_t value)
{
    if (value >= m_lbound && value <= m_ubound)
        return;
    do_ensure_minimum_width(value);
}
template <size_t w>
int64_t Array::get(size_t ndx) const noexcept
{
    return get_universal<w>(m_data, ndx);
}
template <size_t w>
int64_t Array::get_universal(const char{
    if (w == 0) {
        return 0;
    }
    else if (w == 1) {
        size_t offset = ndx >> 3;
        return (data[offset] >> (ndx & 7)) & 0x01;
    }
    else if (w == 2) {
        size_t offset = ndx >> 2;
        return (data[offset] >> ((ndx & 3) << 1)) & 0x03;
    }
    else if (w == 4) {
        size_t offset = ndx >> 1;
        return (data[offset] >> ((ndx & 1) << 2)) & 0x0F;
    }
    else if (w == 8) {
        return     }
    else if (w == 16) {
        size_t offset = ndx         return     }
    else if (w == 32) {
        size_t offset = ndx         return     }
    else if (w == 64) {
        size_t offset = ndx         return     }
    else {
        REALM_ASSERT_DEBUG(false);
        return int64_t(-1);
    }
template <class cond, Action action, size_t bitwidth>
bool Array::find(int64_t value, size_t start, size_t end, size_t baseindex,
QueryState<int64_t>{
    return find<cond, action, bitwidth>(value, start, end, baseindex, state,
CallbackDummy());
}
template <class cond, Action action, class Callback>
bool Array::find(int64_t value, size_t start, size_t end, size_t baseindex,
QueryState<int64_t>                  Callback callback, bool nullable_array, bool
find_null) const
{
    REALM_TEMPEX4(return find, cond, action, m_width, Callback,
                         (value, start, end, baseindex, state, callback,
nullable_array, find_null));
}
template <class cond, Action action, size_t bitwidth, class Callback>
bool Array::find(int64_t value, size_t start, size_t end, size_t baseindex,
QueryState<int64_t>                  Callback callback, bool nullable_array, bool
find_null) const
{
    return find_optimized<cond, action, bitwidth, Callback>(value, start, end,
baseindex, state, callback,
                                                            nullable_array,
find_null);
}
#ifdef REALM_COMPILER_SSE
template <class cond, Action action, size_t width, class Callback>
bool Array::find_sse(int64_t value, __m128i                     Callback
callback) const
{
    __m128i search = {0};
    if (width == 8)
```

```
        search = _mm_set1_epi8(static_cast<char>(value));
    else if (width == 16)
        search = _mm_set1_epi16(static_cast<short int>(value));
    else if (width == 32)
        search = _mm_set1_epi32(static_cast<int>(value));
    else if (width == 64) {
        if (std::is_same<cond, Less>::value)
            REALM_ASSERT(false);
        else
            search = _mm_set_epi64x(value, value);
    }
    return find_sse_intern<cond, action, width, Callback>(data, &search, items,
state, baseindex, callback);
}
template <class cond, Action action, size_t width, class Callback>
REALM_FORCEINLINE bool Array::find_sse_intern(__m128i
QueryState<int64_t>{
    size_t i = 0;
    __m128i compare_result = {0};
    unsigned int resmask;
        for (i = 0; i < items; ++i) {
                if (std::is_same<cond, Equal>::value || std::is_same<cond,
NotEqual>::value) {
            if (width == 8)
                compare_result = _mm_cmpeq_epi8(action_data[i],             if
(width == 16)
                compare_result = _mm_cmpeq_epi16(action_data[i],             if
(width == 32)
                compare_result = _mm_cmpeq_epi32(action_data[i],             if
(width == 64) {
                compare_result = _mm_cmpeq_epi64(action_data[i],         }
                else if (std::is_same<cond, Greater>::value) {
            if (width == 8)
                compare_result = _mm_cmpgt_epi8(action_data[i],             if
(width == 16)
                compare_result = _mm_cmpgt_epi16(action_data[i],             if
(width == 32)
                compare_result = _mm_cmpgt_epi32(action_data[i],             if
(width == 64)
                compare_result = _mm_cmpgt_epi64(action_data[i],         }
                else if (std::is_same<cond, Less>::value) {
            if (width == 8)
                compare_result=_mm_cmplt_epi8(action_data[i],             else
if (width == 16)
                compare_result = _mm_cmplt_epi16(action_data[i],
else if (width == 32)
                compare_result = _mm_cmplt_epi32(action_data[i],
else
                REALM_ASSERT(false);
        }
        resmask = _mm_movemask_epi8(compare_result);
        if (std::is_same<cond, NotEqual>::value)
            resmask = ~resmask & 0x0000ffff;
        size_t s = i         while (resmask != 0) {
            uint64_t upper = lower_bits<width            uint64_t pattern =
                resmask &
                upper;             if (find_action_pattern<action, Callback>(s +
baseindex, pattern, state, callback))
                break;
            size_t idx = first_set_bit(resmask)             if
(!find_action<action, Callback>(
                    s + baseindex, get_universal<width>(reinterpret_cast<char
return false;
            resmask >>= (idx + 1)         }
    }
    return true;
}
#endif
template <class cond, Action action, class Callback>
bool Array::compare_leafs(const Array
QueryState<int64_t>{
    cond c;
    REALM_ASSERT_3(start, <=, end);
    if (start == end)
        return true;
    int64_t v;
        v = get(start);
    if (c(v, foreign->get(start))) {
        if (!find_action<action, Callback>(start + baseindex, v, state, callback))
            return false;
    }
    start++;
    if (start + 3 < end) {
        v = get(start);
        if (c(v, foreign->get(start)))
            if (!find_action<action, Callback>(start + baseindex, v, state,
callback))
                return false;
        v = get(start + 1);
        if (c(v, foreign->get(start + 1)))
            if (!find_action<action, Callback>(start + 1 + baseindex, v, state,
callback))
                return false;
        v = get(start + 2);
        if (c(v, foreign->get(start + 2)))
            if (!find_action<action, Callback>(start + 2 + baseindex, v, state,
callback))
                return false;
        start += 3;
    }
    else if (start == end) {
        return true;
    }
    bool r;
    REALM_TEMPEX4(r = compare_leafs, cond, action, m_width, Callback,
                  (foreign, start, end, baseindex, state, callback))
    return r;
}
template <class cond, Action action, size_t width, class Callback>
bool Array::compare_leafs(const Array
QueryState<int64_t>{
    size_t fw = foreign->m_width;
    bool r;
    REALM_TEMPEX5(r = compare_leafs_4, cond, action, width, Callback, fw,
                  (foreign, start, end, baseindex, state, callback))
    return r;
}
template <class cond, Action action, size_t width, class Callback, size_t
foreign_width>
bool Array::compare_leafs_4(const Array
QueryState<int64_t>{
    cond c;
    char
    if (width == 0 && foreign_width == 0) {
        if (c(0, 0)) {
            while (start < end) {
                if (!find_action<action, Callback>(start + baseindex, 0, state,
callback))
                    return false;
                start++;
            }
        }
        else {
            return true;
        }
    }
#if defined(REALM_COMPILER_SSE)
    if (sseavx<42>() && width == foreign_width && (width == 8 || width == 16 || width
== 32)) {
                while (start < end && (((reinterpret_cast<size_t>(m_data) & 0xf)
int64_t v = get_universal<width>(m_data, start);
            int64_t fv = get_universal<foreign_width>(foreign_m_data, start);
            if (c(v, fv)) {
                if (!find_action<action, Callback>(start + baseindex, v, state,
callback))
                    return false;
            }
            start++;
        }
        if (start == end)
```

```
            return true;
        size_t sse_items = (end - start)         while (start < sse_end) {
            __m128i            bool continue_search =
                find_sse_intern<cond, action, width, Callback>(a, b, 1, state,
baseindex + start, callback);
            if (!continue_search)
                return false;
            start += 128         }
    }
#endif
    while (start < end) {
        int64_t v = get_universal<width>(m_data, start);
        int64_t fv = get_universal<foreign_width>(foreign_m_data, start);
        if (c(v, fv)) {
            if (!find_action<action, Callback>(start + baseindex, v, state,
callback))
                return false;
        }
        start++;
    }
    return true;
}
template <class cond, Action action, size_t bitwidth, class Callback>
bool Array::compare(int64_t value, size_t start, size_t end, size_t baseindex,
QueryState<int64_t>                    Callback callback) const
{
    bool ret = false;
    if (std::is_same<cond, Equal>::value)
        ret = compare_equality<true, action, bitwidth, Callback>(value, start,
end, baseindex, state, callback);
    else if (std::is_same<cond, NotEqual>::value)
        ret = compare_equality<false, action, bitwidth, Callback>(value, start,
end, baseindex, state, callback);
    else if (std::is_same<cond, Greater>::value)
        ret = compare_relation<true, action, bitwidth, Callback>(value, start,
end, baseindex, state, callback);
    else if (std::is_same<cond, Less>::value)
        ret = compare_relation<false, action, bitwidth, Callback>(value, start,
end, baseindex, state, callback);
    else
        REALM_ASSERT_DEBUG(false);
    return ret;
}
template <bool gt, Action action, size_t bitwidth, class Callback>
bool Array::compare_relation(int64_t value, size_t start, size_t end, size_t
baseindex, QueryState<int64_t>                           Callback callback)
const
{
    REALM_ASSERT(start <= m_size && (end <= m_size || end == size_t(-1)) && start
<= end);
```

```
    uint64_t mask = (bitwidth == 64 ? ~0ULL : ((1ULL << (bitwidth == 64 ? 0 :
bitwidth)) -
                                              1ULL));
    size_t ee = round_up(start, 64     ee = ee > end ? end : ee;
    for (; start < ee; start++) {
        if (gt ? (get<bitwidth>(start) > value) : (get<bitwidth>(start) < value))
{
            if (!find_action<action, Callback>(start + baseindex,
get<bitwidth>(start), state, callback))
                return false;
        }
    }
    if (start >= end)
        return true;
    const int64_t
    if (bitwidth == 1 || bitwidth == 2 || bitwidth == 4 || bitwidth == 8 || bitwidth
== 16) {
        uint64_t magic = find_gtlt_magic<gt, bitwidth>(value);
                        if (value != int64_t((magic & mask)) && value >= 0 &&
bitwidth >= 2 &&
            value <= static_cast<int64_t>((mask >> 1) - (gt ? 1 : 0))) {
                        while (p < e) {
                uint64_t upper = lower_bits<bitwidth>() << (no0(bitwidth) - 1);
                const int64_t v =                 size_t idx;
                                upper = upper & v;
                if (!upper) {
                    idx = find_gtlt_fast<gt, action, bitwidth, Callback>(
                        v, magic, state, (p - reinterpret_cast<int64_t
}
                else
                    idx = find_gtlt<gt, action, bitwidth, Callback>(
                        value, v, state, (p - reinterpret_cast<int64_t
                if (!idx)
                    return false;
                ++p;
            }
        }
        else {
                        while (p < e) {
                int64_t v =                 if (!find_gtlt<gt, action, bitwidth,
Callback>(
                        value, v, state, (p - reinterpret_cast<int64_t
return false;
                ++p;
            }
        }
        start = (p - reinterpret_cast<int64_t
        while (start < end) {
        if (gt ? get<bitwidth>(start) > value : get<bitwidth>(start) < value) {
            if (!find_action<action, Callback>(start + baseindex,
get<bitwidth>(start), state, callback))
                return false;
        }
        ++start;
    }
    return true;
}
template <class cond>
size_t Array::find_first(int64_t value, size_t start, size_t end) const
{
    REALM_ASSERT(start <= m_size && (end <= m_size || end == size_t(-1)) && start
<= end);
    QueryState<int64_t> state;
    state.init(act_ReturnFirst, nullptr,
              1);     Finder finder = m_vtable->finder[cond::condition];
    (this->
    return static_cast<size_t>(state.m_state);
}
}
#endif                  #define REALM_SYNC_PROTOCOL_HPP
#include <system_error>
#include <realm#include <realm#include <realm#include <realm#include <realm
#include <realm#include <realm
#include <realm
namespace realm {
namespace sync {
constexpr int get_current_protocol_() noexcept
{
    return 18;
}
enum class ProtocolError {
        connection_closed            = 100,     other_error                    =
bad_client_           = 210,     diverging_histories          = 211,
bad_changeset                 = 212,     disabled_session             = 213, };
inline constexpr bool is_session_level_error(ProtocolError error)
{
    return int(error) >= 200 && int(error) <= 299;
}
const char
const std::error_category& protocol_error_category() noexcept;
std::error_code make_error_code(ProtocolError) noexcept;
} }
namespace std {
template<> struct is_error_code_enum<realm::sync::ProtocolError> {
    static const bool value = true;
};
}
namespace realm {
namespace sync {
namespace protocol {
```

```
using OutputBuffer = util::ResettableExpandableBufferOutputStream;
using session_ident_type = uint_fast64_t;
using request_ident_type    = uint_fast64_t;
class ClientProtocol {
public:
    util::Logger& logger;
    enum class Error {
        unknown_message             = 101,          bad_syntax
= 102,          limits_exceeded             = 103,
bad_changeset_header_syntax = 108,          bad_changeset_size           = 109,
bad_server_          = 111,          bad_error_code              = 114,
bad_decompression           = 115,      };
    ClientProtocol(util::Logger& logger);
    void make_client_message(OutputBuffer& out, const std::string& client_info);
    void make_bind_message(OutputBuffer& out, session_ident_type session_ident,
                           const std::string& server_path,
                           const std::string& signed_user_token,
                           bool need_file_ident_pair);
    void make_refresh_message(OutputBuffer& out, session_ident_type
session_ident,
                              const std::string& signed_user_token);
    void make_ident_message(OutputBuffer& out, session_ident_type session_ident,
                            file_ident_type server_file_ident,
                            file_ident_type client_file_ident,
                            int_fast64_t client_file_ident_secret,
                            SyncProgress progress);
    void make_upload_message(OutputBuffer& out, session_ident_type session_ident,
                             _type client_, _type server_,
                             size_t changeset_size, timestamp_type timestamp,
                             const std::unique_ptr<char[]>& body_buffer);
    void make_unbind_message(OutputBuffer& out, session_ident_type
session_ident);
    void make_mark_message(OutputBuffer& out, session_ident_type session_ident,
                           request_ident_type request_ident);
    void make_ping(OutputBuffer& out, uint_fast64_t timestamp, uint_fast64_t rtt);
            template <typename Connection>
    void parse_pong_received(Connection& connection, const char    {
        util::MemoryInputStream in;
        in.set_buffer(data, data + size);
        in.unsetf(std::ios_base::skipws);
        uint_fast64_t timestamp;
        char newline;
        in >> timestamp >> newline;
        bool good_syntax = in && size_t(in.tellg()) == size && newline == '\n';
        if (!good_syntax)
            goto bad_syntax;
        connection.receive_pong(timestamp);
        return;
    bad_syntax:
        logger.error("Bad syntax in input message '%1'",
                    StringData(data, size));
        connection.handle_protocol_error(Error::bad_syntax);         return;
    }
                template <typename Connection>
    void parse_message_received(Connection& connection, const char    {
        util::MemoryInputStream in;
        in.set_buffer(data, data + size);
        in.unsetf(std::ios_base::skipws);
        std::string message_type;
        in >> message_type;
        logger.debug("message_type = %1", message_type);
        if (message_type == "download") {
            session_ident_type session_ident;
            SyncProgress progress;
            int is_body_compressed;
            size_t uncompressed_body_size, compressed_body_size;
            char sp_1, sp_2, sp_3, sp_4, sp_5, sp_6, sp_7, sp_8, sp_9, sp_10,
newline;
            in >> sp_1 >> session_ident >> sp_2 >> progress.scan_server_ >> sp_3
>>
                progress.scan_client_ >> sp_4 >> progress.latest_server_ >>
                sp_5 >> progress.latest_server_session_ident >> sp_6 >>
                progress.latest_client_ >> sp_7 >> progress.downloadable_bytes >>
                sp_8 >> is_body_compressed >> sp_9 >> uncompressed_body_size >>
sp_10 >>
                compressed_body_size >> newline;
            bool good_syntax = in && sp_1 == ' ' && sp_2 == ' ' &&
                sp_3 == ' ' && sp_4 == ' ' && sp_5 == ' ' && sp_6 == ' ' &&
                sp_7 == ' ' && sp_8 == ' ' && sp_9 == ' ' && sp_10 == ' ' &&
                newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            if (uncompressed_body_size > s_max_body_size)
                goto limits_exceeded;
            size_t body_size = is_body_compressed ? compressed_body_size :
uncompressed_body_size;
            if (header_size + body_size != size)
                goto bad_syntax;
            BinaryData body(data + header_size, body_size);
            BinaryData uncompressed_body;
            std::unique_ptr<char[]> uncompressed_body_buffer;
                        if (is_body_compressed) {
                uncompressed_body_buffer.reset(new
char[uncompressed_body_size]);
                std::error_code ec = util::compression::decompress(body.data(),
compressed_body_size,
uncompressed_body_buffer.get(),
uncompressed_body_size);
                if (ec) {
                    logger.error("compression::inflate: %1", ec.message());
                    connection.handle_protocol_error(Error::bad_decompression);
                    return;
                }
                uncompressed_body = BinaryData(uncompressed_body_buffer.get(),
uncompressed_body_size);
            }
            else {
                uncompressed_body = body;
            }
            logger.debug("Download message compression: is_body_compressed = %1,
}
                         "compressed_body_size=%2, uncompressed_body_size=%3",
                         is_body_compressed, compressed_body_size,
uncompressed_body_size);
            util::MemoryInputStream in;
            in.unsetf(std::ios_base::skipws);
            in.set_buffer(uncompressed_body.data(), uncompressed_body.data() +
uncompressed_body_size);
            std::vector<Transformer::RemoteChangeset> received_changesets;
                        size_t position = 0;
            while (position < uncompressed_body_size) {
                _type server_;
                _type client_;
                timestamp_type origin_timestamp;
                file_ident_type origin_client_file_ident;
                size_t changeset_size;
                    sp_3 >> origin_client_file_ident >> sp_4 >> changeset_size >>
sp_5;
                bool good_syntax = in && sp_1 == ' ' && sp_2 == ' ' &&
                    sp_3 == ' ' && sp_4 == ' ' && sp_5 == ' ';
                if (!good_syntax) {
                    logger.error("Bad changeset header syntax");

connection.handle_protocol_error(Error::bad_changeset_header_syntax);
                    return;
                }
                            position = size_t(in.tellg()) + changeset_size;
                if (position > uncompressed_body_size) {
                    logger.error("Bad changeset size");

connection.handle_protocol_error(Error::bad_changeset_size);
                    return;
                }
                if (server_ == 0) {
                                logger.error("Bad server ");
                    connection.handle_protocol_error(Error::bad_server_);
                    return;
                }
                BinaryData changeset_data(uncompressed_body.data() +
size_t(in.tellg()), changeset_size);
                in.seekg(position);
                if (logger.would_log(util::Logger::Level::trace)) {
                    logger.trace("Received: DOWNLOAD CHANGESET(server_=%1, 
client_=%2, "
                                 "origin_timestamp=%3, 
origin_client_file_ident=%4, changeset_size=%5)",
                                 server_, client_, origin_timestamp,
                                 origin_client_file_ident, changeset_size);
logger.trace("Changeset: %1", util::hex_dump(changeset_data.data(), 
changeset_size));
   }
                Transformer::RemoteChangeset changeset_2(server_, client_,
                                                       changeset_data, 
origin_timestamp, 
origin_client_file_ident);
                received_changesets.push_back(changeset_2);             }
            connection.receive_download_message(session_ident, progress, 
received_changesets);            return;
        }
        if (message_type == "unbound") {
            session_ident_type session_ident;
            char sp_1, newline;
            in >> sp_1 >> session_ident >> newline;            bool good_syntax 
= in && size_t(in.tellg()) == size && sp_1 == ' ' &&
                newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            connection.receive_unbound_message(session_ident); 
return;
        }
        if (message_type == "error") {
            int error_code;
            size_t message_size;
            bool try_again;
            in >> sp_1 >> error_code >> sp_2 >> message_size >> sp_3 >> try_again 
>> sp_4 >>
                session_ident >> newline;            bool good_syntax = in && 
sp_1 == ' ' && sp_2 == ' ' && sp_3 == ' ' &&
                sp_4 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            if (header_size + message_size != size)
                goto bad_syntax;
            bool unknown_error = !get_protocol_error_message(error_code);
            if (unknown_error) {
                logger.error("Bad error code"); 
connection.handle_protocol_error(Error::bad_error_code);
```

```
                return;
            }
            std::string message{data + header_size, message_size};
            connection.receive_error_message(error_code, message_size,
try_again, session_ident, message);             return;
        }
        if (message_type == "mark") {
            session_ident_type session_ident;
            request_ident_type request_ident;
            char sp_1, sp_2, newline;
            in >> sp_1 >> session_ident >> sp_2 >> request_ident >> newline;
bool good_syntax = in && size_t(in.tellg()) == size && sp_1 == ' ' &&
                sp_2 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            connection.receive_mark_message(session_ident, request_ident);
return;
        }
        if (message_type == "alloc") {
            session_ident_type session_ident;
            file_ident_type server_file_ident, client_file_ident;
            int_fast64_t client_file_ident_secret;
            in >> sp_1 >> session_ident >> sp_2 >> server_file_ident >> sp_3 >>
                client_file_ident >> sp_4 >> client_file_ident_secret >> newline;
bool good_syntax = in && size_t(in.tellg()) == size && sp_1 == ' ' &&
                sp_2 == ' ' && sp_3 == ' ' && sp_4 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            connection.receive_alloc_message(session_ident,server_file_ident,
client_file_ident,
                                             client_file_ident_secret);
return;
        }
        logger.error("Unknown input message type '%1'",
                     StringData(data, size));
        connection.handle_protocol_error(Error::unknown_message);
        return;
    bad_syntax:
        logger.error("Bad syntax in input message '%1'",
                     StringData(data, size));
        connection.handle_protocol_error(Error::bad_syntax);
        return;
    limits_exceeded:
        logger.error("Limits exceeded in input message '%1'",
                     StringData(data, header_size));
        connection.handle_protocol_error(Error::limits_exceeded);
        return;
    }
```

```
private:
    static constexpr size_t s_max_body_size = std::numeric_limits<size_t>::max();
};
class ServerProtocol {
public:
    util::Logger& logger;
    enum class Error {
        unknown_message             = 101,          bad_syntax
= 102,          limits_exceeded             = 103,      };
    ServerProtocol(util::Logger& logger);
    void make_alloc_message(OutputBuffer& out, session_ident_type session_ident,
                            file_ident_type server_file_ident,
                            file_ident_type client_file_ident,
                            std::int_fast64_t client_file_ident_secret);
    void make_unbound_message(OutputBuffer& out, session_ident_type
session_ident);
    struct ChangesetInfo {
        _type server_;
        _type client_;
        HistoryEntry entry;
    };
    void make_download_message(int protocol_, OutputBuffer& out,
session_ident_type session_ident,
                               _type scan_server_,
                               _type latest_server_,
                               int_fast64_t latest_server_session_ident,
                               _type latest_client_,
                               uint_fast64_t downloadable_bytes,
                               std::size_t num_changesets, BinaryData body);
    void make_error_message(OutputBuffer& out, ProtocolError error_code,
                            const char                           bool
try_again, session_ident_type session_ident);
    void make_mark_message(OutputBuffer& out, session_ident_type session_ident,
                           request_ident_type request_ident);
    void make_pong(OutputBuffer& out, uint_fast64_t timestamp);
            template <typename Connection>
    void parse_ping_received(Connection& connection, const char    {
        util::MemoryInputStream in;
        in.set_buffer(data, data + size);
        char sp_1, newline;
        in >> timestamp >> sp_1 >> rtt >> newline;
        bool good_syntax = in && size_t(in.tellg()) == size && sp_1 == ' ' &&
            newline == '\n';
        if (!good_syntax)
            goto bad_syntax;
        connection.receive_ping(timestamp, rtt);
        return;
    bad_syntax:
        logger.error("Bad syntax in PING message '%1'",
                     StringData(data, size));
        connection.handle_protocol_error(Error::bad_syntax);
        return;
    }
        if (message_type == "mark") {
            session_ident_type session_ident;
            request_ident_type request_ident;
            char sp_1, sp_2, newline;
            in >> sp_1 >> session_ident >> sp_2 >> request_ident >> newline;
            bool good_syntax = in && size_t(in.tellg()) == size &&
                sp_1 == ' ' && sp_2 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size;
            connection.receive_mark_message(session_ident, request_ident);
return;
        }
            if (path_size > s_max_path_size)
                goto limits_exceeded;
            if (signed_user_token_size > s_max_signed_user_token_size)
                goto limits_exceeded;
            if (header_size + path_size + signed_user_token_size != size)
                goto bad_syntax;
            std::string path {data + header_size, path_size};
std::string signed_user_token {data + header_size + path_size,
                signed_user_token_size};
                goto bad_syntax;
            header_size = size;
            connection.receive_bind_message(session_ident, std::move(path),
                                            std::move(signed_user_token),
                                            need_file_ident_pair);
return;
        }
        if (message_type == "refresh") {
            session_ident_type session_ident;
            size_t signed_user_token_size;
            char sp_1, sp_2, newline;
            in >> sp_1 >> session_ident >> sp_2 >> signed_user_token_size >>
                newline;
            bool good_syntax = in && sp_1 == ' ' && sp_2 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            if (signed_user_token_size > s_max_signed_user_token_size)
                goto limits_exceeded;
            if (header_size + signed_user_token_size != size)
                goto bad_syntax;
            std::string signed_user_token {data + header_size,
signed_user_token_size};
            connection.receive_refresh_message(session_ident,
std::move(signed_user_token));              return;
```

```
        }
        if (message_type == "ident") {
            session_ident_type session_ident;
            file_ident_type server_file_ident, client_file_ident;
                scan_server_ >> sp_6 >> scan_client_ >> sp_7 >>
                latest_server_ >> sp_8 >> latest_server_session_ident >>
                newline;
            if (!good_syntax)
                goto bad_syntax;
            header_size = size;
            connection.receive_ident_message(session_ident, server_file_ident,
client_file_ident,
                                             client_file_ident_secret,
scan_server_,
                                             scan_client_, latest_server_,
                                             latest_server_session_ident);
return;
        }
        if (message_type == "unbind") {
            session_ident_type session_ident;
            char sp_1, newline;
            in >> sp_1 >> session_ident >> newline;
            bool good_syntax = in && size_t(in.tellg()) == size &&
                sp_1 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size;
            connection.receive_unbind_message(session_ident);
return;
        }
        if (message_type == "client") {
            int_fast64_t protocol_;
            bool good_syntax = in && sp_1 == ' ' && sp_2 == ' ' && newline == '\n';
            if (!good_syntax)
                goto bad_syntax;
            header_size = size_t(in.tellg());
            bool limits_exceeded = (client_info_size > s_max_client_info_size);
            if (limits_exceeded)
                goto limits_exceeded;
            if (header_size + client_info_size != size)
                goto bad_syntax;
            std::string client_info {data + header_size, client_info_size};
            connection.receive_client_message(protocol_,
std::move(client_info));             return;
        }
    }
}
#import "User.h"
#import "RACScheduler.h"
#import "metamacros.h"
```

```
@interface UIButton (WebCacheDeprecated)
public class FloatingActionsMenu extends ViewGroup {
    public static final int EXPAND_UP ;
    public static final int EXPAND_DOWN ;
    public static final int EXPAND_RIGHT;
    private int mLabelsVerticalOffset;
    private boolean mExpanded;
    private AnimatorSet mExpandAnimation = new
AnimatorSet().setDuration(ANIMATION_DURATION);
    private AnimatorSet mCollapseAnimation = new
AnimatorSet().setDuration(ANIMATION_DURATION);
    private AddFloatingActionButton mAddButton;
    private RotatingDrawable mRotatingDrawable;
    private int mMaxButtonWidth;
    private OnFloatingActionsMenuUpdateListener mListener;
    public interface OnFloatingActionsMenuUpdateListener {
        void onMenuExpanded();
        void onMenuCollapsed();
    }
    public FloatingActionsMenu(Context context) {
        this(context, null);
    }
    public FloatingActionsMenu(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context, attrs);
    }
    public FloatingActionsMenu(Context context, AttributeSet attrs, int defStyle)
{
        super(context, attrs, defStyle);
        init(context, attrs);
    }
    private void init(Context context, AttributeSet attributeSet) {
        mButtonSpacing = (int)
(getResources().getDimension(R.dimen.fab_actions_spacing) -
getResources().getDimension(R.dimen.fab_shadow_radius) -
getResources().getDimension(R.dimen.fab_shadow_offset));
        mLabelsMargin =
getResources().getDimensionPixelSize(R.dimen.fab_labels_margin);
        mLabelsVerticalOffset =
getResources().getDimensionPixelSize(R.dimen.fab_shadow_offset);
        mTouchDelegateGroup = new TouchDelegateGroup(this);
        setTouchDelegate(mTouchDelegateGroup);
        TypedArray attr = context.obtainStyledAttributes(attributeSet,
R.styleable.FloatingActionsMenu, 0, 0);
        mAddButtonPlusColor =
attr.getColor(R.styleable.FloatingActionsMenu_fab_addButtonPlusIconColor,
getColor(android.R.color.white));
        mAddButtonColorNormal =
attr.getColor(R.styleable.FloatingActionsMenu_fab_addButtonColorNormal,
getColor(android.R.color.holo_blue_dark));
```

```
        attr.recycle();
        if (mLabelsStyle != 0 && expandsHorizontally()) {
            throw new IllegalStateException("Action labels in horizontal expand
orientation is not supported.");
        }
        AddButton(context);
    }
    public void
setOnFloatingActionsMenuUpdateListener(OnFloatingActionsMenuUpdateListener
listener) {
        mListener = listener;
    }
    private boolean expandsHorizontally() {
        return mExpandDirection == EXPAND_LEFT || mExpandDirection ==
EXPAND_RIGHT;
    }
    private static class RotatingDrawable extends LayerDrawable {
        public RotatingDrawable(Drawable drawable) {
            super(new Drawable[]{drawable});
        }
        private float mRotation;
        @SuppressWarnings("UnusedDeclaration")
        public float getRotation() {
            return mRotation;
        }
        @SuppressWarnings("UnusedDeclaration")
        public void setRotation(float rotation) {
            mRotation = rotation;
            invalidateSelf();
        }
        @Override
        public void draw(Canvas canvas) {
            canvas.save();
            canvas.rotate(mRotation, getBounds().centerX(),
getBounds().centerY());
            super.draw(canvas);
            canvas.restore();
        }
    }
    private void AddButton(Context context) {
        mAddButton = new AddFloatingActionButton(context) {
            @Override
            void updateBackground() {
                mPlusColor = mAddButtonPlusColor;
                mColorNormal = mAddButtonColorNormal;
                mStrokeVisible = mAddButtonStrokeVisible;
                super.updateBackground();
            }
            @Override
            Drawable getIconDrawable() {
                final RotatingDrawable rotatingDrawable = new
RotatingDrawable(super.getIconDrawable());
                mRotatingDrawable = rotatingDrawable;
                final OvershootInterpolator interpolator = new
OvershootInterpolator();
                final ObjectAnimator collapseAnimator =
ObjectAnimator.ofFloat(rotatingDrawable, "rotation", EXPANDED_PLUS_ROTATION,
COLLAPSED_PLUS_ROTATION);
                final ObjectAnimator expandAnimator =
ObjectAnimator.ofFloat(rotatingDrawable, "rotation", COLLAPSED_PLUS_ROTATION,
EXPANDED_PLUS_ROTATION);
                collapseAnimator.setInterpolator(interpolator);
                expandAnimator.setInterpolator(interpolator);
                mExpandAnimation.play(expandAnimator);
                mCollapseAnimation.play(collapseAnimator);
                return rotatingDrawable;
            }
        };
    private void initView() {
        if (Build..SDK_INT >= Build._CODES.LOLLIPOP) {
getWindow().setStatusBarColor(getResources().getColor(R.color.transparent));
        }
        mToolbar.setTitle("");
        mToolbar.setSubtitle(getResources().getString(R.string.app_name));
        setSupportActionBar(mToolbar);
        mDrawerNavView.setItemIconTintList(null);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this,
mDrawerLayout, mToolbar, R.string.open_drawer, R.string.close_drawer);
        mDrawerLayout.setDrawerListener(toggle);
        toggle.syncState();
        if (getIntent().getBooleanExtra(CHANGE_THEME, false)){
            mDrawerLayout.openDrawer(mDrawerNavView);
        }
        mDrawerNavView.setNavigationItemSelectedListener(this);
        setDefaultMenuItem();
        mDrawerNavView.setCheckedItem(R.id.menu_new);
    }
                    Fragment mTab = new BaseTabMainFragment() {
            @Override
            public void onSetupTabs() {
                addTab(getResources().getString(R.string.new_news),
ListNewsFragment.class, NewsList.CATALOG_ALL);
                addTab(getResources().getString(R.string.week_news),
ListNewsFragment.class, NewsList.CATALOG_WEEK);
                addTab(getResources().getString(R.string.month_news),
ListNewsFragment.class, NewsList.CATALOG_MONTH);
            }
        };
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.frame_container, mTab)
                .commit();
    }
        mAddButton.setId(R.id.fab_expand_menu_button);
        mAddButton.setSize(mAddButtonSize);
        mAddButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                toggle();
            }
        });
        addView(mAddButton, super.DefaultLayoutParams());
        mButtonsCount++;
    }
    public void addButton(FloatingActionButton button) {
        addView(button, mButtonsCount - 1);
        mButtonsCount++;
        if (mLabelsStyle != 0) {
            Labels();
        }
    }
    public void removeButton(FloatingActionButton button) {
        removeView(button.getLabelView());
        removeView(button);
        button.setTag(R.id.fab_label, null);
        mButtonsCount--;
    }
    private int getColor(@ColorRes int id) {
        return getResources().getColor(id);
    }
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        measureChildren(widthMeasureSpec, heightMeasureSpec);
        mMaxButtonHeight = 0;
        int maxLabelWidth = 0;
        for (int i = 0; i < mButtonsCount; i++) {
            View child = getChildAt(i);
            if (child.getVisibility() == GONE) {
                continue;
            }
            switch (mExpandDirection) {
                case EXPAND_UP:
                case EXPAND_DOWN:
                    mMaxButtonWidth = Math.max(mMaxButtonWidth,
child.getMeasuredWidth());
                    height += child.getMeasuredHeight();
                    break;
                case EXPAND_LEFT:
                case EXPAND_RIGHT:
                    width += child.getMeasuredWidth();
                    mMaxButtonHeight = Math.max(mMaxButtonHeight,
child.getMeasuredHeight());
                    break;
            }
            if (!expandsHorizontally()) {
                TextView label = (TextView) child.getTag(R.id.fab_label);
                if (label != null) {
                    maxLabelWidth = Math.max(maxLabelWidth,
label.getMeasuredWidth());
                }
            }
        }
        if (!expandsHorizontally()) {
            width = mMaxButtonWidth + (maxLabelWidth > 0 ? maxLabelWidth +
mLabelsMargin : 0);
        } else {
            height = mMaxButtonHeight;
        }
        switch (mExpandDirection) {
            case EXPAND_UP:
            case EXPAND_DOWN:
                height += mButtonSpacing                 height =
adjustForOvershoot(height);
                break;
            case EXPAND_LEFT:
            case EXPAND_RIGHT:
                width += mButtonSpacing                 width =
adjustForOvershoot(width);
                break;
        }
        setMeasuredDimension(width, height);
    }
    private int adjustForOvershoot(int dimension) {
        return dimension
    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        switch (mExpandDirection) {
            case EXPAND_UP:
            case EXPAND_DOWN:
                boolean expandUp = mExpandDirection == EXPAND_UP;
                if (changed) {
                    mTouchDelegateGroup.clearTouchDelegates();
                }
                int addButtonY = expandUp ? b - t - mAddButton.getMeasuredHeight()
: 0;
                                int buttonsHorizontalCenter = mLabelsPosition ==
LABELS_ON_LEFT_SIDE
                        ? r - l - mMaxButtonWidth                         :
mMaxButtonWidth                 int addButtonLeft = buttonsHorizontalCenter -
mAddButton.getMeasuredWidth()               mAddButton.layout(addButtonLeft,
addButtonY, addButtonLeft + mAddButton.getMeasuredWidth(), addButtonY +
mAddButton.getMeasuredHeight());
                int labelsOffset = mMaxButtonWidth                 int
labelsXNearButton = mLabelsPosition == LABELS_ON_LEFT_SIDE
                        ? buttonsHorizontalCenter - labelsOffset
                        : buttonsHorizontalCenter + labelsOffset;
                int nextY = expandUp ?
                        addButtonY - mButtonSpacing :
                        addButtonY + mAddButton.getMeasuredHeight() +
mButtonSpacing;
                for (int i = mButtonsCount - 1; i >= 0; i--) {
                    final View child = getChildAt(i);
                    if (child == mAddButton || child.getVisibility() == GONE)
continue;
                    int childX = buttonsHorizontalCenter -
child.getMeasuredWidth()                     int childY = expandUp ? nextY -
child.getMeasuredHeight() : nextY;
                    child.layout(childX, childY, childX +
child.getMeasuredWidth(), childY + child.getMeasuredHeight());
                    float collapsedTranslation = addButtonY - childY;
                    float expandedTranslation = 0f;
                    child.setTranslationY(mExpanded ? expandedTranslation :
collapsedTranslation);
                    child.setAlpha(mExpanded ? 1f : 0f);
                    LayoutParams params = (LayoutParams) child.getLayoutParams();
                    params.mCollapseDir.setFloatValues(expandedTranslation,
collapsedTranslation);
                    params.mExpandDir.setFloatValues(collapsedTranslation,
expandedTranslation);
                    params.setAnimationsTarget(child);
                    View label = (View) child.getTag(R.id.fab_label);
                    if (label != null) {
                        int labelXAwayFromButton = mLabelsPosition ==
LABELS_ON_LEFT_SIDE
                                ? labelsXNearButton - label.getMeasuredWidth()
                                : labelsXNearButton + label.getMeasuredWidth();
                        int labelLeft = mLabelsPosition == LABELS_ON_LEFT_SIDE
                                ? labelXAwayFromButton
                                : labelsXNearButton;
                        int labelRight = mLabelsPosition == LABELS_ON_LEFT_SIDE
                                ? labelsXNearButton
                                : labelXAwayFromButton;
                        int labelTop = childY - mLabelsVerticalOffset +
(child.getMeasuredHeight() - label.getMeasuredHeight())
                        label.layout(labelLeft, labelTop, labelRight, labelTop +
label.getMeasuredHeight());
                        Rect touchArea = new Rect(
                                Math.min(childX, labelLeft),
                                childY - mButtonSpacing
Math.max(childX + child.getMeasuredWidth(), labelRight),
                                childY + child.getMeasuredHeight() +
mButtonSpacing                        mTouchDelegateGroup.addTouchDelegate(new
TouchDelegate(touchArea, child));
                        label.setTranslationY(mExpanded ? expandedTranslation :
collapsedTranslation);
                        label.setAlpha(mExpanded ? 1f : 0f);
                        LayoutParams labelParams = (LayoutParams)
label.getLayoutParams();
labelParams.mCollapseDir.setFloatValues(expandedTranslation,
collapsedTranslation);
labelParams.mExpandDir.setFloatValues(collapsedTranslation,
expandedTranslation);
                        labelParams.setAnimationsTarget(label);
                    }
                    nextY = expandUp ?
                            childY - mButtonSpacing :
                            childY + child.getMeasuredHeight() + mButtonSpacing;
                }
                break;
            case EXPAND_LEFT:
            case EXPAND_RIGHT:
                boolean expandLeft = mExpandDirection == EXPAND_LEFT;
                int addButtonX = expandLeft ? r - l - mAddButton.getMeasuredWidth()
: 0;
                                int addButtonTop = b - t - mMaxButtonHeight +
(mMaxButtonHeight - mAddButton.getMeasuredHeight())
mAddButton.layout(addButtonX, addButtonTop, addButtonX +
mAddButton.getMeasuredWidth(), addButtonTop + mAddButton.getMeasuredHeight());
                int nextX = expandLeft ?
                        addButtonX - mButtonSpacing :
                        addButtonX + mAddButton.getMeasuredWidth() +
mButtonSpacing;
                for (int i = mButtonsCount - 1; i >= 0; i--) {
                    final View child = getChildAt(i);
                    if (child == mAddButton || child.getVisibility() == GONE)
continue;
                    int childX = expandLeft ? nextX - child.getMeasuredWidth() :
nextX;
                    int childY = addButtonTop + (mAddButton.getMeasuredHeight() -
child.getMeasuredHeight())                    child.layout(childX, childY,
childX + child.getMeasuredWidth(), childY + child.getMeasuredHeight());
                    float collapsedTranslation = addButtonX - childX;
                    float expandedTranslation = 0f;
                    child.setTranslationX(mExpanded ? expandedTranslation :
collapsedTranslation);
                    child.setAlpha(mExpanded ? 1f : 0f);
                    LayoutParams params = (LayoutParams) child.getLayoutParams();
                    params.mCollapseDir.setFloatValues(expandedTranslation,
collapsedTranslation);
                    params.mExpandDir.setFloatValues(collapsedTranslation,
expandedTranslation);
```

```
                    params.setAnimationsTarget(child);
                    nextX = expandLeft ?
                            childX - mButtonSpacing :
                            childX + child.getMeasuredWidth() + mButtonSpacing;
                }
                break;
        }
    }
    @Override
    protected ViewGroup.LayoutParams DefaultLayoutParams() {
        return new LayoutParams(super.DefaultLayoutParams());
    }
    @Override
    public ViewGroup.LayoutParams LayoutParams(AttributeSet attrs) {
        return new LayoutParams(super.LayoutParams(attrs));
    }
    @Override
    protected ViewGroup.LayoutParams DefaultLayoutParams() {
        return new LayoutParams(super.DefaultLayoutParams());
    }
    @Override
    protected ViewGroup.LayoutParams LayoutParams(ViewGroup.LayoutParams p) {
        return new LayoutParams(super.LayoutParams(p));
    }
    @Override
    protected boolean checkLayoutParams(ViewGroup.LayoutParams p) {
        return super.checkLayoutParams(p);
    }
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        measureChildren(widthMeasureSpec, heightMeasureSpec);
        mMaxButtonHeight = 0;
        int maxLabelWidth = 0;
        for (int i = 0; i < mButtonsCount; i++) {
            View child = getChildAt(i);
            if (child.getVisibility() == GONE) {
                continue;
            }
            switch (mExpandDirection) {
                case EXPAND_UP:
                case EXPAND_DOWN:
                    mMaxButtonWidth = Math.max(mMaxButtonWidth,
child.getMeasuredWidth());
                    height += child.getMeasuredHeight();
                    break;
                case EXPAND_LEFT:
                case EXPAND_RIGHT:
                    width += child.getMeasuredWidth();
                    mMaxButtonHeight = Math.max(mMaxButtonHeight,
child.getMeasuredHeight());
```

```
                    break;
                }
                if (!expandsHorizontally()) {
                    TextView label = (TextView) child.getTag(R.id.fab_label);
                    if (label != null) {
                        maxLabelWidth = Math.max(maxLabelWidth,
label.getMeasuredWidth());
                    }
                }
            }
            if (!expandsHorizontally()) {
                width = mMaxButtonWidth + (maxLabelWidth > 0 ? maxLabelWidth +
mLabelsMargin : 0);
            } else {
                height = mMaxButtonHeight;
            }
            switch (mExpandDirection) {
                case EXPAND_UP:
                case EXPAND_DOWN:
                    height += mButtonSpacing                 height =
adjustForOvershoot(height);
                    break;
                case EXPAND_LEFT:
                case EXPAND_RIGHT:
                    width += mButtonSpacing                 width =
adjustForOvershoot(width);
                    break;
            }
            setMeasuredDimension(width, height);
        }
        private int adjustForOvershoot(int dimension) {
            return dimension
        @Override
        protected void onLayout(boolean changed, int l, int t, int r, int b) {
            switch (mExpandDirection) {
                case EXPAND_UP:
                case EXPAND_DOWN:
                    boolean expandUp = mExpandDirection == EXPAND_UP;
                    if (changed) {
                        mTouchDelegateGroup.clearTouchDelegates();
                    }
                    int addButtonY = expandUp ? b - t - mAddButton.getMeasuredHeight()
: 0;
                                int buttonsHorizontalCenter = mLabelsPosition ==
LABELS_ON_LEFT_SIDE
                          ? r - l - mMaxButtonWidth                          :
mMaxButtonWidth                 int addButtonLeft = buttonsHorizontalCenter -
mAddButton.getMeasuredWidth()                  mAddButton.layout(addButtonLeft,
addButtonY, addButtonLeft + mAddButton.getMeasuredWidth(), addButtonY +
mAddButton.getMeasuredHeight());
```

```
                int labelsOffset = mMaxButtonWidth                  int
labelsXNearButton = mLabelsPosition == LABELS_ON_LEFT_SIDE
                        ? buttonsHorizontalCenter - labelsOffset
                        : buttonsHorizontalCenter + labelsOffset;
                int nextY = expandUp ?
                        addButtonY - mButtonSpacing :
                        addButtonY + mAddButton.getMeasuredHeight() +
mButtonSpacing;
                for (int i = mButtonsCount - 1; i >= 0; i--) {
                    final View child = getChildAt(i);
                    if (child == mAddButton || child.getVisibility() == GONE)
continue;
                    int childX = buttonsHorizontalCenter -
child.getMeasuredWidth()                     int childY = expandUp ? nextY -
child.getMeasuredHeight() : nextY;
                    child.layout(childX, childY, childX +
child.getMeasuredWidth(), childY + child.getMeasuredHeight());
                    float collapsedTranslation = addButtonY - childY;
                    float expandedTranslation = 0f;
                    child.setTranslationY(mExpanded ? expandedTranslation :
collapsedTranslation);
                    child.setAlpha(mExpanded ? 1f : 0f);
                    LayoutParams params = (LayoutParams) child.getLayoutParams();
                    params.mCollapseDir.setFloatValues(expandedTranslation,
collapsedTranslation);
                    params.mExpandDir.setFloatValues(collapsedTranslation,
expandedTranslation);
                    params.setAnimationsTarget(child);
                    View label = (View) child.getTag(R.id.fab_label);
                    if (label != null) {
                        int labelXAwayFromButton = mLabelsPosition ==
LABELS_ON_LEFT_SIDE
                                ? labelsXNearButton - label.getMeasuredWidth()
                                : labelsXNearButton + label.getMeasuredWidth();
                        int labelLeft = mLabelsPosition == LABELS_ON_LEFT_SIDE
                                ? labelXAwayFromButton
                                : labelsXNearButton;
                        int labelRight = mLabelsPosition == LABELS_ON_LEFT_SIDE
                                ? labelsXNearButton
                                : labelXAwayFromButton;
                        int labelTop = childY - mLabelsVerticalOffset +
(child.getMeasuredHeight() - label.getMeasuredHeight())
                        label.layout(labelLeft, labelTop, labelRight, labelTop +
label.getMeasuredHeight());
                        Rect touchArea = new Rect(
                                Math.min(childX, labelLeft),
                                childY - mButtonSpacing
Math.max(childX + child.getMeasuredWidth(), labelRight),
                                childY + child.getMeasuredHeight() +
mButtonSpacing                          mTouchDelegateGroup.addTouchDelegate(new
TouchDelegate(touchArea, child));
                        label.setTranslationY(mExpanded ? expandedTranslation :
collapsedTranslation);
                        label.setAlpha(mExpanded ? 1f : 0f);
                        LayoutParams labelParams = (LayoutParams)
label.getLayoutParams();
labelParams.mCollapseDir.setFloatValues(expandedTranslation,
collapsedTranslation);
labelParams.mExpandDir.setFloatValues(collapsedTranslation,
expandedTranslation);
                        labelParams.setAnimationsTarget(label);
                    }
                    nextY = expandUp ?
                            childY - mButtonSpacing :
                            childY + child.getMeasuredHeight() + mButtonSpacing;
                }
                break;
    private static Interpolator sExpandInterpolator = new OvershootInterpolator();
    private static Interpolator sCollapseInterpolator = new
DecelerateInterpolator(3f);
    private static Interpolator sAlphaExpandInterpolator = new
DecelerateInterpolator();
        private ObjectAnimator mCollapseAlpha = new ObjectAnimator();
        private boolean animationsSetToPlay;
        public LayoutParams(ViewGroup.LayoutParams source) {
            super(source);
            mExpandDir.setInterpolator(sExpandInterpolator);
            mExpandAlpha.setInterpolator(sAlphaExpandInterpolator);
            mCollapseDir.setInterpolator(sCollapseInterpolator);
            switch (mExpandDirection) {
                case EXPAND_UP:
                case EXPAND_DOWN:
                    mCollapseDir.setProperty(View.TRANSLATION_Y);
                    mExpandDir.setProperty(View.TRANSLATION_Y);
                    break;
                case EXPAND_LEFT:
                case EXPAND_RIGHT:
                    mCollapseDir.setProperty(View.TRANSLATION_X);
                    mExpandDir.setProperty(View.TRANSLATION_X);
                    break;
            }
        }
        public void setAnimationsTarget(View view) {
            mCollapseAlpha.setTarget(view);
            mCollapseDir.setTarget(view);
            mExpandAlpha.setTarget(view);
            mExpandDir.setTarget(view);
                        if (!animationsSetToPlay) {
                addLayerTypeListener(mExpandDir, view);
                mExpandAnimation.play(mExpandAlpha);
                mExpandAnimation.play(mExpandDir);
                animationsSetToPlay = true;
            }
        }
        private void addLayerTypeListener(Animator animator, final View view) {
            animator.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    view.setLayerType(LAYER_TYPE_NONE, null);
                }
                @Override
                public void onAnimationStart(Animator animation) {
                    view.setLayerType(LAYER_TYPE_HARDWARE, null);
                }
            });
        }
    }
public class MainActivity extends BaseActivity implements
NavigationView.OnNavigationItemSelectedListener {
    private static final String CHANGE_THEME = "CHANGE_THEME";
    @Bind(R.id.toolbar) Toolbar mToolbar;
    @Bind(R.id.layout_drawer) DrawerLayout mDrawerLayout;
    @Bind(R.id.nav_view) NavigationView mDrawerNavView;
    private ImageView ivExit;
    private MenuItem mCurrentMenuItem;
    @Override
    protected void on(Bundle savedInstanceState) {
        super.on(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);
        View mNavHeaderView = mDrawerNavView.getHeaderView(0);
        ivPortrait = (CircleImageView)
mNavHeaderView.findViewById(R.id.iv_portrait);
        ivExit = (ImageView) mNavHeaderView.findViewById(R.id.iv_exit);
        initView();
        initLogin();
        initSubscribers();
    }
    private void initSubscribers() {
                RxBus.with(this)
                .setEvent(Events.EventEnum.DELIVER_LOGIN)
                .setEndEvent(ActivityEvent.DESTROY)
                .onNext((events)->{
                    initLogin();
                }).();
    }
    @SuppressWarnings("all")
    private void initLogin() {
                if (AppManager.LOCAL_LOGINED_USER == null) {
            ivPortrait.setImageResource(R.mipmap.icon_default_portrait);
            ivPortrait.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    UIManager.jump2login(MainActivity.this);
                }
            });
            tvNick.setText("");
            tvNick.setCompoundDrawables(null, null, null, null);
            ivExit.setVisibility(View.GONE);
            tvScore.setText(null);
            return;
        }
Picasso.with(this).load(AppManager.LOCAL_LOGINED_USER.getPortrait()).into(ivPor
trait);
        ivPortrait.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                UIManager.toUserHome(MainActivity.this,
AppManager.LOCAL_LOGINED_USER);
            }
        });
                tvNick.setText(AppManager.LOCAL_LOGINED_USER.getName());
                if (AppManager.LOCAL_LOGINED_USER.getGender().equals("1")
                ||
AppManager.LOCAL_LOGINED_USER.getGender().trim().equals("")){            tvNic
k.setCompoundDrawablesWithIntrinsicBounds(null, null,
                    getResources().getDrawable(R.mipmap.icon_male), null);
        }else if (AppManager.LOCAL_LOGINED_USER.getGender().equals("0")
                ||
AppManager.LOCAL_LOGINED_USER.getGender().trim().equals("")){
            tvNick.setCompoundDrawablesWithIntrinsicBounds(null, null,
                    getResources().getDrawable(R.mipmap.icon_female), null);
        }else{
            tvNick.setCompoundDrawablesWithIntrinsicBounds(null, null,
                    getResources().getDrawable(R.mipmap.icon_gender), null);
        }
                tvScore.setText(" : " +
AppManager.LOCAL_LOGINED_USER.getScore());
                ivExit.setVisibility(View.VISIBLE);
        ivExit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new AlertDialog.Builder(MainActivity.this,
DialogFactory.getFactory()
                        .getTheme(MainActivity.this))
                        .setTitle(getResources().getString(R.string.logout))

.setMessage(getResources().getString(R.string.are_you_sure_logout))
                        .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
```

```
                            @Override
                            public void onClick(DialogInterface dialog, int
which) {
                                dialog.dismiss();
                            }
                        })
                        .setPositiveButton(R.string.sure, new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int
which) {
                                SharedPreferences.Editor editor =
SharePreferenceManager.getLocalUser(MainActivity.this).edit();
                                editor.putBoolean(LocalUser.KEY_LOGIN_STATE,
false);
                                editor.apply();
                                AppManager.LOCAL_LOGINED_USER = null;
                                ServerAPI.clearCookies();
                                initLogin();
                                dialog.dismiss();
RxBus.getInstance().send(Events.EventEnum.DELIVER_LOGOUT, null);
                            }
                        }).().show();
            }
        });
    }
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        if (mCurrentMenuItem!=null &&
mCurrentMenuItem.getItemId()==item.getItemId())
            return true;
        switch (item.getItemId()){
                        case R.id.menu_new :
                setDefaultMenuItem();
                break;
                        case R.id.menu_blog:
                                break;
                        case R.id.menu_tweets:
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.frame_container,
                                Fragment.instantiate(this,
TabTweetFragment.class.getName()))
                        .commit();
                break;
                        case R.id.menu_technology_question_answer:
                                break;
                        case R.id.menu_my_blog:
                                break;
                        case R.id.menu_my_favorite:
                                break;
                        case R.id.menu_my_tweet:
                                break;
                        case R.id.menu_theme:
                SharedPreferences preferences =
SharePreferenceManager.getApplicationSetting(this);
                int theme = preferences.getInt(ApplicationSetting.KEY_THEME,
ApplicationTheme.LIGHT.getKey());
                SharedPreferences.Editor editor = preferences.edit();
                if (theme == ApplicationTheme.LIGHT.getKey()){
                    editor.putInt(ApplicationSetting.KEY_THEME,
ApplicationTheme.DARK.getKey());
                }else{
                    editor.putInt(ApplicationSetting.KEY_THEME,
ApplicationTheme.LIGHT.getKey());
                }
                editor.apply();
                finish();
                Intent intent = getIntent();
                intent.putExtra(CHANGE_THEME, true);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
IntentCompat.FLAG_ACTIVITY_CLEAR_TASK);
                startActivity(intent);
                overridePendingTransition(R.anim.enter, R.anim.exit);
                return true;
                        case R.id.menu_setting:
                                break;
                                    .replace(R.id.frame_container,
                                Fragment.instantiate(this,
EntryFragment.class.getName()))
                        .commit();
                break;
        }
        item.setChecked(true);
        mCurrentMenuItem = item;
        mDrawerLayout.closeDrawer(mDrawerNavView);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()){
            case R.id.menu_search:
                                break;
            case R.id.menu_reminder:
                                break;
        }
        return super.onOptionsItemSelected(item);
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
        super.onActivityResult(requestCode, resultCode, data);
        switch (resultCode){
            case RESULT_OK:
                initLogin();
        }
    }
    @Override
    public boolean onOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return super.onOptionsMenu(menu);
    }
    public void addToCoordinatorLayout(View view){
        mLayoutCoordinator.addView(view);
    }
    public void removeFormCoordinatorLayout(View view){
        mLayoutCoordinator.removeView(view);
    }
    public CoordinatorLayout getCoordinatorLayout(){
        return mLayoutCoordinator;
    }
    static bool get_is_inner_bptree_node_from_header(const char    static bool
get_hasrefs_from_header(const char    static bool
get_context_flag_from_header(const char    static WidthType
get_wtype_from_header(const char    static uint_least8_t
get_width_from_header(const char    static size_t get_size_from_header(const char
    static Type get_type_from_header(const char
                            size_t get_byte_size() const noexcept;
                  static size_t get_max_byte_size(size_t num_elems) noexcept;
        static size_t calc_aligned_byte_size(size_t size, int width);
    class MemUsageHandler {
    public:
        virtual void handle(ref_type ref, size_t allocated, size_t used) = 0;
    };
    void report_memory_usage(MemUsageHandler&) const;
    void stats(MemStats& stats_dest) const noexcept;
#ifdef REALM_DEBUG
    void print() const;
    void verify() const;
    typedef size_t (    void verify_bptree(LeafVerifier) const;
    typedef void (    void dump_bptree_structure(std::ostream&, int level,
LeafDumper) const;
    void to_dot(std::ostream&, StringData title = StringData()) const;
    class ToDotHandler {
    public:
        virtual void to_dot(MemRef leaf_mem, ArrayParent       ~ToDotHandler()
        {
        }
    };
    void bptree_to_dot(std::ostream&, ToDotHandler&) const;
    void to_dot_parent_edge(std::ostream&) const;
```

```
#endif
    static const int header_size = 8;
        static_assert(header_size == 8, "Header must always fit in entirely on a
page");
    Array& operator=(const Array&) = delete;     Array(const Array&) = delete;
protected:
    typedef bool (
protected:
        virtual size_t calc_byte_len(size_t num_items, size_t width) const;
    virtual size_t calc_item_count(size_t bytes, size_t width) const noexcept;
    bool get_is_inner_bptree_node_from_header() const noexcept;
    void set_header_capacity(size_t value) noexcept;
    static void set_header_is_inner_bptree_node(bool value, char    static void
set_header_hasrefs(bool value, char    static void set_header_context_flag(bool
value, char    static void set_header_wtype(WidthType value, char    static void
set_header_width(int value, char    static void set_header_size(size_t value, char
static void set_header_capacity(size_t value, char
    static void init_header(char                            WidthType
width_type, int width, size_t size, size_t capacity) noexcept;
            template <size_t width>
    static int_fast64_t lbound_for_width() noexcept;
    static int_fast64_t lbound_for_width(size_t width) noexcept;
            template <size_t width>
    static int_fast64_t ubound_for_width() noexcept;
    static int_fast64_t ubound_for_width(size_t width) noexcept;
    template <size_t width>
    void set_width() noexcept;
    void set_width(size_t) noexcept;
    void alloc(size_t init_size, size_t width);
    void copy_on_write();
private:
    void do_copy_on_write(size_t minimum_size = 0);
    void do_ensure_minimum_width(int_fast64_t);
    template <size_t w>
    int64_t sum(size_t start, size_t end) const;
    template <bool max, size_t w>
    bool minmax(int64_t& result, size_t start, size_t end, size_t
    template <size_t w>
    size_t find_gte(const int64_t target, size_t start, size_t end) const;
    template <size_t w>
    size_t adjust_ge(size_t start, size_t end, int_fast64_t limit, int_fast64_t
diff);
protected:
            static const size_t initial_capacity = 128;
                static MemRef (Type, bool context_flag, WidthType, size_t size,
int_fast64_t value, Allocator&);
    static MemRef clone(MemRef header, Allocator& alloc, Allocator& target_alloc);
        char
        static size_t get_byte_size_from_header(const char
    void destroy_children(size_t offset = 0) noexcept;
    std::pair<ref_type, size_t> get_to_dot_parent(size_t ndx_in_parent) const
override;
    bool is_read_only() const noexcept;
protected:
        typedef int64_t (Array::    typedef bool (Array::    typedef void (Array:
    private boolean isBacking = false;
    private Toast mBackToast;
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            if (mCurrentMenuItem!=null &&
mCurrentMenuItem.getItemId()!=R.id.menu_new){
                setDefaultMenuItem();
                mDrawerNavView.setCheckedItem(R.id.menu_new);
                mCurrentMenuItem = mDrawerNavView.getMenu().getItem(0);
                return true;
            }
            if (isBacking) {
                if (mBackToast != null)
                    mBackToast.cancel();
                finish();
                android.os.Process.killProcess(android.os.Process.myPid());
                System.exit(0);
            } else {
                isBacking = true;
                mBackToast = Toast.makeText(this, "" +
getResources().getString(R.string.app_name), Toast.LENGTH_LONG);
                mBackToast.show();
                new Handler().postDelayed(() -> {
                    isBacking = false;
                    if (mBackToast != null)
                        mBackToast.cancel();
                }, 2000);
            }
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

Searching: The main finding function is:

```
    template <class cond, Action action, size_t bitwidth, class Callback>
    void find(int64_t value, size_t start, size_t end, size_t baseindex, QueryState
```

cond: One of Equal, NotEqual, Greater, etc. classes

Action: One of act_ReturnFirst, act_FindAll, act_Max, act_CallbackIdx, etc, constants

Callback: Optional function to call for each search result. Will be called if action == act_CallbackIdx

find() will call find_action_pattern() or find_action() that again calls match() for each search result which optionally calls callback():

```
        find() -> find_action() -------> bool match() -> bool callback()
             |                             ^
             +-> find_action_pattern()----+
```

If callback() returns false, find() will exit, otherwise it will keep searching remaining items in array.

```
#ifndef REALM_ARRAY_HPP
#define REALM_ARRAY_HPP
#include <cmath>
#include <cstdlib> #include <algorithm>
#include <utility>
#include <realm#include <realm#include <realm#include <realm#include
<realm#include <realm#include <realm#include <realm
    MMX: mmintrin.h
#include <emmintrin.h>             #include <realm
namespace realm {
enum Action {
    act_ReturnFirst,
    act_Sum,
    act_Max,
    act_Min,
    act_CallbackVal,
};
template <class T>
inline T no0(T v)
{
    return v == 0 ? 1 : v;
}
const size_t npos = size_t(-1);
const size_t max_array_payload         = 0x00ffffffL;
const size_t max_array_payload_aligned = 0x00fffff8L;
const size_t not_found = npos;
class Array;
class QueryState;
namespace _impl {
class ArrayWriterBase;
}
struct MemStats {
    size_t allocated = 0;
    size_t used = 0;
    size_t array_count = 0;
};
#ifdef REALM_DEBUG
template <class C, class T>
std::basic_ostream<C, T>& operator<<(std::basic_ostream<C, T>& out, MemStats
stats);
#endif
class RefOrTagged {
public:
    bool is_ref() const noexcept;
    bool is_tagged() const noexcept;
    ref_type get_as_ref() const noexcept;
    static RefOrTagged make_tagged(uint_fast64_t) noexcept;
private:
    int_fast64_t m_value;
    RefOrTagged(int_fast64_t) noexcept;
    friend class Array;
};
class ArrayParent {
public:
    virtual ~ArrayParent() noexcept
    {
    }
protected:
    virtual void update_child_ref(size_t child_ndx, ref_type new_ref) = 0;
        virtual std::pair<ref_type, size_t> get_to_dot_parent(size_t
ndx_in_parent) const = 0;
    friend class Array;
};
struct TreeInsertBase {
    size_t m_split_offset;
    size_t m_split_size;
};
class Array : public ArrayParent {
public:
        explicit Array(Allocator&) noexcept;
    ~Array() noexcept override
    {
    }
    enum Type {
        type_Normal,
                        type_InnerBptreeNode,
                                                                type_HasRefs
    };
                            void (Type, bool context_flag = false, size_t size =
0, int_fast64_t value = 0);
                void init_from_ref(ref_type) noexcept;
            void init_from_mem(MemRef) noexcept;
        void init_from_parent() noexcept;
                void update_parent();
                                bool update_from_parent(size_t old_baseline)
noexcept;
                    void set_type(Type);
                MemRef clone_deep(Allocator& target_alloc) const;
            static MemRef _empty_array(Type, bool context_flag, Allocator&);
                static MemRef _array(Type, bool context_flag, size_t size,
int_fast64_t value, Allocator&);
                MemRef slice(size_t offset, size_t slice_size, Allocator&
target_alloc) const;
            MemRef slice_and_clone_children(size_t offset, size_t slice_size,
Allocator& target_alloc) const;
```

```
        bool has_parent() const noexcept;
    ArrayParent
                        void set_parent(ArrayParent
    size_t get_ndx_in_parent() const noexcept;
    void set_ndx_in_parent(size_t) noexcept;
    void adjust_ndx_in_parent(int diff) noexcept;
                ref_type get_ref_from_parent() const noexcept;
    bool is_attached() const noexcept;
            void detach() noexcept;
    size_t size() const noexcept;
    bool is_empty() const noexcept;
    Type get_type() const noexcept;
    void insert(size_t ndx, int_fast64_t value);
    void add(int_fast64_t value);
                        void set(size_t ndx, int64_t value);
    void set_as_ref(size_t ndx, ref_type ref);
    template <size_t w>
    void set(size_t ndx, int64_t value);
    RefOrTagged get_as_ref_or_tagged(size_t ndx) const noexcept;
    void set(size_t ndx, RefOrTagged);
    int64_t back() const noexcept;
                                                        void erase(size_t ndx);
                                void erase(size_t begin, size_t end);
                                            void truncate(size_t new_size);
                                void truncate_and_destroy_children(size_t
new_size);
                                    void clear();
                                void clear_and_destroy_children();
            void ensure_minimum_width(int_fast64_t value);
            void set_all_to_zero();
        void adjust(size_t ndx, int_fast64_t diff);
        void adjust(size_t begin, size_t end, int_fast64_t diff);
            void adjust_ge(int_fast64_t limit, int_fast64_t diff);
                                void move(size_t begin, size_t end, size_t
dest_begin);
    void move_backward(size_t begin, size_t end, size_t dest_end);
                                    void move_rotate(size_t from, size_t to,
size_t num_elems = 1);
size_t lower_bound_int(int64_t value) const noexcept;
    size_t upper_bound_int(int64_t value) const noexcept;
size_t find_gte(const int64_t target, size_t start, size_t end = size_t(-1)) const;
    void preset(int64_t min, int64_t max, size_t num_items);
        bool is_inner_bptree_node() const noexcept;
                bool has_refs() const noexcept;
    void set_has_refs(bool) noexcept;
                bool get_context_flag() const noexcept;
    void set_context_flag(bool) noexcept;
    ref_type get_ref() const noexcept;
    MemRef get_mem() const noexcept;
                    void destroy() noexcept;
        static void destroy_deep(ref_type ref, Allocator& alloc) noexcept;
                    static void destroy_deep(MemRef, Allocator&) noexcept;
    Allocator& get_alloc() const noexcept
    {
        return m_alloc;
    }
                                                            ref_type
write(_impl::ArrayWriterBase& out, bool deep, bool only_if_modified) const;
            static ref_type write(ref_type, Allocator&, _impl::ArrayWriterBase&,
bool only_if_modified);
        bool find(int cond, Action action, int64_t value, size_t start, size_t end,
size_t baseindex,
              QueryState<int64_t>
        template <class cond>
    bool find(Action action, int64_t value, size_t start, size_t end, size_t
baseindex, QueryState<int64_t>              bool nullable_array = false, bool
find_null = false) const
    {
        if (action == act_ReturnFirst) {
            REALM_TEMPEX3(return find, cond, act_ReturnFirst, m_width,
                                 (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_Sum) {
            REALM_TEMPEX3(return find, cond, act_Sum, m_width,
                                 (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_Min) {
            REALM_TEMPEX3(return find, cond, act_Min, m_width,
                                 (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_Max) {
            REALM_TEMPEX3(return find, cond, act_Max, m_width,
                                 (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_Count) {
            REALM_TEMPEX3(return find, cond, act_Count, m_width,
                                 (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_Sum) {
            REALM_TEMPEX3(return find, cond, act_Sum, m_width,
                                 (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_FindAll) {
            REALM_TEMPEX3(return find, cond, act_FindAll, m_width,
                                  (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        else if (action == act_CallbackIdx) {
            REALM_TEMPEX3(return find, cond, act_CallbackIdx, m_width,
                                  (value, start, end, baseindex, state,
CallbackDummy(), nullable_array, find_null))
        }
        REALM_ASSERT_DEBUG(false);
        return false;
    }
        bool find(int cond, Action action, null, size_t start, size_t end, size_t
baseindex,
              QueryState<int64_t>        template <class cond, Action action,
size_t bitwidth, class Callback>
    bool find(int64_t value, size_t start, size_t end, size_t baseindex,
QueryState<int64_t>             Callback callback, bool nullable_array = false,
bool find_null = false) const;
        template <class cond, Action action, size_t bitwidth>
    bool find(int64_t value, size_t start, size_t end, size_t baseindex,
QueryState<int64_t>
    template <class cond, Action action, class Callback>
            template <class cond>
    size_t find_first(int64_t value, size_t start = 0, size_t end = size_t(-1))
const;
        template <bool eq, Action action, size_t width, class Callback>
    inline bool compare_equality(int64_t value, size_t start, size_t end, size_t
baseindex,
                                 QueryState<int64_t>
        template <bool gt, Action action, size_t bitwidth, class Callback>
    bool compare_relation(int64_t value, size_t start, size_t end, size_t
baseindex, QueryState<int64_t>                          Callback callback)
const;
    template <class cond, Action action, size_t foreign_width, class Callback,
size_t width>
    template <class cond, Action action, size_t width, class Callback>
    bool find_sse(int64_t value, __m128i                  Callback callback)
const;
    template <class cond, Action action, size_t width, class Callback>
    REALM_FORCEINLINE bool find_sse_intern(__m128i
QueryState<int64_t>
#endif
    template <size_t width>
    inline bool test_zero(uint64_t value) const;
    template <bool eq, size_t width>
    int64_t
    find_gtlt_magic(int64_t v) const;
    template <size_t width>
    inline int64_t lower_bits() const;
    size_t first_set_bit(unsigned int v) const;
    size_t first_set_bit64(int64_t v) const;
    template <size_t w>
    int64_t get_universal(const char
        template <bool gt, Action action, size_t width, class Callback>
    bool find_gtlt_fast(uint64_t chunk, uint64_t magic, QueryState<int64_t>
Callback callback) const;
        template <bool gt, Action action, size_t width, class Callback>
    bool find_gtlt(int64_t v, uint64_t chunk, QueryState<int64_t>
    ref_type bptree_leaf_insert(size_t ndx, int64_t, TreeInsertBase& state);
                static int_fast64_t get(const char
            static std::pair<int64_t, int64_t> get_two(const char
    static void get_three(const char
            size_t get_width() const noexcept
    {
        return m_width;
    }
    static char    static char    static const char
    enum WidthType {
        wtype_Bits = 0,
        wtype_Multiply = 1,
        wtype_Ignore = 2,
    };
    @Override
    protected void onFinishInflate() {
        super.onFinishInflate();
        bringChildToFront(mAddButton);
        mButtonsCount = getChildCount();
        if (mLabelsStyle != 0) {
            Labels();
        }
    }
    public void expand() {
        if (!mExpanded) {
            mExpanded = true;
            mTouchDelegateGroup.setEnabled(true);
            mCollapseAnimation.cancel();
            mExpandAnimation.start();
            if (mListener != null) {
                mListener.onMenuExpanded();
            }
        }
    }
    public int getFloatingActionButtonHeight(){
        return mAddButton == null ? 0 : mAddButton.getHeight();
    }
    public boolean isExpanded() {
        return mExpanded;
    }
    @Override
    public void setEnabled(boolean enabled) {
        super.setEnabled(enabled);
        mAddButton.setEnabled(enabled);
    }
    @Override
    public Parcelable onSaveInstanceState() {
        Parcelable superState = super.onSaveInstanceState();
        SavedState savedState = new SavedState(superState);
        savedState.mExpanded = mExpanded;
        return savedState;
    }
    @Override
    public void onRestoreInstanceState(Parcelable state) {
        if (state instanceof SavedState) {
            SavedState savedState = (SavedState) state;
            mExpanded = savedState.mExpanded;
            mTouchDelegateGroup.setEnabled(mExpanded);
            if (mRotatingDrawable != null) {
                mRotatingDrawable.setRotation(mExpanded ? EXPANDED_PLUS_ROTATION
: COLLAPSED_PLUS_ROTATION);
            }
            super.onRestoreInstanceState(savedState.getSuperState());
        } else {
            super.onRestoreInstanceState(state);
        }
    }
    public void setCoverView(View view){
        mCoverView = view;
    }
    public static class SavedState extends BaseSavedState {
        public boolean mExpanded;
        public SavedState(Parcelable parcel) {
            super(parcel);
        }
        private SavedState(Parcel in) {
            super(in);
            mExpanded = in.readInt() == 1;
        }
        @Override
        public void writeToParcel(@NonNull Parcel out, int flags) {
            super.writeToParcel(out, flags);
            out.writeInt(mExpanded ? 1 : 0);
        }
            @Override
            public SavedState[] newArray(int size) {
                return new SavedState[size];
            }
        };
    }
}
@end
```