\libraries\app\application.cpp

```
#include <graphene/app/api.hpp>
#include <graphene/app/api_access.hpp>
#include <graphene/app/application.hpp>
#include <graphene/app/plugin.hpp>

#include <graphene/chain/genesis_state.hpp>
#include <graphene/chain/protocol/fee_schedule.hpp>
#include <graphene/chain/protocol/types.hpp>

#include <graphene/egenesis/egenesis.hpp>

#include <graphene/net/core_messages.hpp>
#include <graphene/net/exceptions.hpp>

#include <graphene/utilities/key_conversion.hpp>
#include <graphene/chain/worker_evaluator.hpp>

#include <fc/smart_ref_impl.hpp>

#include <fc/asio.hpp>
#include <fc/io/fstream.hpp>
#include <fc/rpc/api_connection.hpp>
#include <fc/rpc/websocket_api.hpp>
#include <fc/network/resolve.hpp>
#include <fc/crypto/base64.hpp>

#include <boost/filesystem/path.hpp>
#include <boost/signals2.hpp>
#include <boost/range/algorithm/reverse.hpp>
#include <boost/algorithm/string.hpp>

#include <iostream>

#include <fc/log/file_appender.hpp>
#include <fc/log/logger.hpp>
#include <fc/log/logger_config.hpp>

#include <boost/range/adaptor/reversed.hpp>

namespace graphene { namespace app {
    using net::item_hash_t;
    using net::item_id;
    using net::message;
    using net::block_message;
    using net::trx_message;

    using chain::block_header;
    using chain::signed_block_header;
    using chain::signed_block;
    using chain::block_id_type;

    using std::vector;
    namespace bpo = boost::program_options;
```

```
namespace detail {
        graphene::chain::genesis_state_type create_example_genesis() {.... }
}
}}
#include "application_impl.hxx"
namespace graphene { namespace app { namespace detail {
void application_impl::reset_p2p_node(const fc::path& data_dir)
std::vector<fc::ip::endpoint> application_impl::resolve_string_to_ip_endpoints(const std::string& endpoint_string)
void application_impl::new_connection( const fc::http::websocket_connection_ptr& c )
void application_impl::reset_websocket_server()
void application_impl::reset_websocket_tls_server()
void application_impl::set_dbg_init_key( graphene::chain::genesis_state_type& genesis, const std::string& init_key )
void application_impl::startup()
optional< api_access_info > application_impl::get_api_access_info(const string& username)const
void application_impl::set_api_access_info(const string& username, api_access_info&& permissions)
bool application_impl::has_item(const net::item_id& id)
bool application_impl::handle_block(const graphene::net::block_message& blk_msg, bool sync_mode,
                    std::vector<fc::uint160_t>& contained_transaction_message_ids)
void application_impl::handle_transaction(const graphene::net::trx_message& transaction_message)
void application_impl::handle_message(const message& message_to_process)
bool application_impl::is_included_block(const block_id_type& block_id)
std::vector<item_hash_t> application_impl::get_block_ids(const std::vector<item_hash_t>& blockchain_synopsis,
                                    uint32_t& remaining_item_count,
                                    uint32_t limit)
message application_impl::get_item(const item_id& id)
chain_id_type application_impl::get_chain_id() const
std::vector<item_hash_t> application_impl::get_blockchain_synopsis(const item_hash_t& reference_point,
                                                uint32_t number_of_blocks_after_reference_point)
void application_impl::sync_status(uint32_t item_type, uint32_t item_count)
void application_impl::connection_count_changed(uint32_t c)
uint32_t application_impl::get_block_number(const item_hash_t& block_id)
fc::time_point_sec application_impl::get_block_time(const item_hash_t& block_id)
item_hash_t application_impl::get_head_block_id() const
uint32_t application_impl::estimate_last_known_fork_from_git_revision_timestamp(uint32_t unix_timestamp) const
void application_impl::error_encountered(const std::string& message, const fc::oexception& error)
uint8_t application_impl::get_current_block_interval_in_seconds() const
} } } // namespace graphene namespace app namespace detail

namespace graphene { namespace app {
application::application() : my(new detail::application_impl(this))
application::~application()
void application::set_program_options(boost::program_options::options_description& command_line_options,
                    boost::program_options::options_description& configuration_file_options) const
void application::initialize(const fc::path& data_dir, const boost::program_options::variables_map& options)
void application::startup()
std::shared_ptr<abstract_plugin> application::get_plugin(const string& name) const
net::node_ptr application::p2p_node()
std::shared_ptr<chain::database> application::chain_database() const
void application::set_block_production(bool producing_blocks)
optional< api_access_info > application::get_api_access_info( const string& username )const
void application::set_api_access_info(const string& username, api_access_info&& permissions)
bool application::is_finished_syncing() const
void graphene::app::application::enable_plugin(const string& name)
```

			void graphene::app::application::**add_available_plugin**(std::shared_ptr<graphene::app::abstract_plugin> p)
			void application::**shutdown_plugins**()
			void application::**shutdown**()
			void application::**:initialize_plugins**(const boost::program_options::variables_map& options)
			void application::**startup_plugins**()
			const application_options& application::**:get_options**()
		}} // namespace detail	
			7/9/2018
			BitShares Core Release 2.0.180612