Описание структур интерфейса взаимодействия

|  |  |  |
| --- | --- | --- |
| Структура | Данные структуры | Описание |
| Point |  | Станционный путь или перегон, описанный внутри поездной нитки |
|  | std::string name | Имена тех объектов, которые поезд должен посетить в том же порядке, в котором он их должен посетить. Обе конечные станции также здесь указаны. Кодировка UTF-8. |
|  | unsigned int dt | Сколько секунд поезд должен находиться на объекте (заказ времени на погрузку-разгрузку и т.д.) |
|  | double lenght | Длина поезда [метры] на объекте (если отличается от предыдущего участка, то изменение произошло на этом участке) |
|  | double weight | Вес поезда [тонн] (если отличается от предыдущего участка,то изменение произошло на этом участке) |
|  | int maxspeed | Максимальная скорость на этом участке |
|  | bool needOnlyThisWay | Если указан станционный путь, то ДА, если нужен именно этот путь или НЕТ если подойдет любой станционный путь |
| Nitka |  | Нитка, возвращаемая нитка должна быть описана полностью, входная - только ключевые станционные пути и/или перегоны |
|  | unsigned int StartTime | Желаемое время подачи поезда на первый объект (то есть объект points[0]), секунды с 00:00 1 января 1970г. |
|  | std::string type | Тип |
|  | int priority | Приоритет |
|  | std::vector<Point> points | Массив станционных путей или перегонов, может быть как полным, так и включать только ключевые точки |
| NitkaID : public Nitka |  | Нитка исполненного движения |
|  | unsigned int idStartTime | Время начала исполненного движения |
|  | std::vector<Point> idPoints | Исполненное движение по маршруту points |
|  | double idLastPosition | Положение на последнем (idPoints.back()) участке [0-1] |
|  | unsigned int pgStartTime | Время начала прогнозного движения |
|  | std::vector<Point> pgPoints | Прогнозное движение по маршруту points |
| Zapret |  | Аншлаг, структура может хранить или аншлаг с полным запретом движения, или аншлаг ограничения по скорости |
|  | std::string name | Имя объекта |
|  | int from, to | С какого по какой момент времени включительно запрещено посещение объекта. Если запрет постоянный, то from=-1 to=-1 |
|  | int maxspeed | Максимальная скорость (если <= 0 запрет движения, > 0 ограничения скорости) |
| Line |  | ребро направленного графа ЖД |
|  | enum LineDirection : int  {  Unknown = -1,  Direction0,  Direction1,  }; | Возможные направления ребра графа ЖД |
|  | std::string srcNode | Название начальной ноды графа |
|  | std::string dstNode | Название конечной ноды графа |
|  | LineDirection srcDirection | Направление начальной ноды графа |
|  | LineDirection dstDirection | Направление конечной ноды графа |
| StandardTime |  | Стандартное время прохождения ноды конкретным типом поезда |
|  | std::string type | Тип поезда |
|  | int time | Время прохождения в прямом направлении |
|  | int timeRev | Время прохождения в обратном направлении |
| Node |  | Вершина направленного графа ЖД, станционный путь или перегон. |
|  | std::string name | Имя объекта |
|  | int dist\_size | Размер в метрах |
|  | std::vector<StandardTime> standard\_times | Массив стандарных времен прохождения этой ноды |
| Graph |  | Граф ЖД |
|  | std::vector<Node> nodes | Вершины графа |
|  | std::vector<Line> lines | Ребра графа |
| Config |  | Настройки алгоритмов |
|  | int Do1 | Интервал времени времени [сек] освобождения пути перед поездом |
|  | int Posle1 | Интервал времени времени [сек] освобождения пути после поезда |
|  | std::vector<int> NumVar1 | Число выдаваемых вариантов по каждому поезду |
|  | int nvar1 | Общее число выдаваемых вариантов |

#### Описание функций интерфейса взаимодействия

//! возвращает название библиотеки

extern "C" LIBRARY\_API const char\* aimName();

//! возвращает версию библиотеки

extern "C" LIBRARY\_API const char\* aimVersion();

//! возвращает версию протокола

extern "C" LIBRARY\_API int aimProtocolVersion();

/\*!

\* инициализация алгоритма, производится один раз при первом запуске библиотеки

\* @param[in] data сериализованные данные (см serializeInit)

\* @return сериализованные данные (см serializeInitResult)

\*/

extern "C" LIBRARY\_API const char\* aimInit(const char\* data);

/\*!

\* запуск работы в однопоточном режиме, используется только для построения расписания

\* @param[in] data сериализованные данные (см serializeWork)

\* @return сериализованные данные (см serializeWorkResult)

\*/

extern "C" LIBRARY\_API const char\* aimWork(const char\* data);

/\*!

\* запуск работы в многопоточном режиме

\* @param[in] data сериализованные данные (см serializeStartWork)

\* @param[in] \*dataready обратный вызов для функции, сигнализирующей о готовности данных для демонстрации результата

\* @return сериализованные данные (см serializeStartWorkResult)

\*/

extern "C" LIBRARY\_API const char\* aimStartWork(const char\* data, void(\*dataready)(void));

/\*!

\* получение результата работы библиотеки

\* @param[in] data параметр зарезервирован для будущих версий

\* @return сериализованные данные (см serializeGetResult)

\*/

extern "C" LIBRARY\_API const char\* aimGetResult(const char\* data);

// указание о завершении работы библиотеки

extern "C" LIBRARY\_API void aimClose();

#### Описание функций индексатора протокола сериализации интерфейса взаимодействия

/\*! \* индексатор версии протокола и типа функции сериализуемых данных

\*/

class CVersion {

public:

static const int s\_str\_size = 4; //! длинна строки типа функции сериализуемых данных

static const int s\_version\_size = 7;//! длинна строки версии протокола

CVersion(); //! конструктор класса CVersion, выставляет дефолтную версию протокола.

//! конструктор класса CVersion, выставляет версию протокола.

CVersion(const std::array<char, s\_str\_size>& magic, int ver);

//! конструктор класса CVersion, заполняет из данных индексатора тип функции и версию

CVersion(const std::string& magic, int ver);

//! заполняет из данных индексатора тип функции и версию

void fromData(const std::array<char, s\_version\_size>& magic);

//! возвращет истину, если содержимое начальных байтов нулевое

bool empty() const;

//! возвращет константный указатель на данные индексатора

const char\* data() const;

//! возвращет указатель на данные индексатора

char\* data();

//! возвращет данные индексатора

const std::array<char, s\_version\_size>& magic() const;

int version() const;

protected:

//! хранит 7 байт 4 первых байта - тип функции 3 последних - версия

std::array<char, s\_version\_size> m\_magic;

// версия протокола отдельно, совпадает с тем что лежит в m\_magic

int m\_version;

};

#### Описание функций сериализатора и десериализатора интерфейса взаимодействия

  class CSerializer {

public:

     /\*!

     \* сериализация данных инициализации алгоритмов

     \* происходит на стороне СУР

     \* @param[in] statefname название файла конфигурации для конкретного алгоритма

     \* @param[in] cfg      общие настройки для всех алгоритмов

     \* @param[in] Graph    граф ЖД

     \* @return указатель на сериализованные данные

     \*/

        const char\* serializeInit(const std::string & statefname, const Config& cfg, const Graph & graph\_input);

        /\*!

     \* сериализация рабочих данных алгоритмов для использования в однопоточном режиме, используется только для построения расписания

     \* происходит на стороне СУР

     \* @param[in] zakaz          список ниток

     \* @param[in] zapr           список аншлагов

     \* @param[in] currentTime  текущее время

     \* @param[in] isPrognoz      устаревший параметр, не используется

     \* @return указатель на сериализованные данные

     \*/

        const char\* serializeWork(const std::vector <Nitka> & zakaz, const std::vector<Zapret> & zapr, int currentTime, bool isPrognoz);

        /\*!

     \* сериализация рабочих данных алгоритмов для использования в многопоточном режиме

     \* происходит на стороне СУР

     \* @param[in] zakaz          список ниток и исполненного движения

     \* @param[in] zapr           список аншлагов

     \* @param[in] currentTime  текущее время

     \* @param[in] interval максимальный интервал прогнозирования, в секундах

        \* @param[in] isPrognoz      ключ режима работы алгоритма

     \* @return указатель на сериализованные данные

     \*/

     const char\* serializeStartWork(const std::vector <NitkaID> & zakaz, const std::vector<Zapret> & zapr, int currentTime, int interval, bool isPrognoz);

        /\*!

     \* сериализация результата инициализации алгоритмов

     \* происходит на стороне интеллектуального модуля

     \* @param[in] messages сообщения

     \* @return указатель на сериализованные данные

    \*/

        const char\* serializeInitResult(const std::string & messages);

        /\*!

     \* сериализация результата запуска алгоритмов для использования в однопоточном режиме, используется только для построения расписания

     \* происходит на стороне интеллектуального модуля

     \* @param[in] messages       сообщения

     \* @param[in] result                списки ниток, варианты результата работы алгоритма

     \* @return указатель на сериализованные данные

     \*/

     const char\* serializeWorkResult(const std::string & messages, const std::vector <std::vector <Nitka> > & result);

        /\*!

     \* сериализация  результата запуска алгоритмов для использования в многопоточном режиме

     \* происходит на стороне интеллектуального модуля

     \* @param[in] messages       сообщения

     \* @return указатель на сериализованные данные

     \*/

     const char\* serializeStartWorkResult(const std::string & messages);

        /\*!

     \* сериализация результата работы алгоритмов

     \* происходит на стороне интеллектуального модуля

     \* @param[in] messages       сообщения

     \* @param[in] progress       прогресс выполнения задачи [0-1]

     \* @param[in] result                списки ниток, варианты результата работы алгоритма

     \* @return указатель на сериализованные данные

     \*/

     const char\* serializeGetResult(const std::string & messages, double progress, const std::vector <std::vector <NitkaID> > & result);

        /\*!

     \* десериализация данных инициализации алгоритмов

     \* происходит на стороне интеллектуального модуля

     \* @param[in] statefname название файла конфигурации для конкретного алгоритма

     \* @param[in] cfg      общие настройки для всех алгоритмов

     \* @param[in] Graph    граф ЖД

     \* @return указатель на сериализованные данные

     \*/

        void deserializeInit(const char\* data, std::string& statefname, Config& cfg, Graph& graph\_input);

        /\*!

     \* десериализация рабочих данных алгоритмов для использования в однопоточном режиме, используется только для построения расписания

     \* происходит на стороне интеллектуального модуля

     \* @param[in] zakaz          список ниток

     \* @param[in] zapr           список аншлагов

     \* @param[in] currentTime  текущее время

     \* @param[in] isPrognoz      устаревший параметр, не используется

     \* @return указатель на сериализованные данные

     \*/

     void deserializeWork(const char\* data, std::vector <Nitka> & zakaz, std::vector<Zapret> & zapr, int& currentTime, bool& isPrognoz);

        /\*!

     \* десериализация рабочих данных алгоритмов для использования в многопоточном режиме

     \* происходит на стороне интеллектуального модуля

     \* @param[in] zakaz          список ниток и исполненного движения

     \* @param[in] zapr           список аншлагов

     \* @param[in] currentTime  текущее время

     \* @param[in] interval максимальный интервал прогнозирования, в секундах

     \* @param[in] isPrognoz      ключ режима работы алгоритма

     \* @return указатель на сериализованные данные

     \*/

     void deserializeStartWork(const char\* data, std::vector <NitkaID> & zakaz, std::vector<Zapret> & zapr, int& currentTime, int& interval, bool& isPrognoz);

        /\*!

     \* десериализация результата инициализации алгоритмов

     \* происходит на стороне СУР

     \* @param[in] messages сообщения

     \* @return указатель на сериализованные данные

     \*/

        void deserializeInitResult(const char\* data, std::string & messages);

        /\*!

     \* десериализация результата запуска алгоритмов для использования в однопоточном режиме, используется только для построения расписания

     \* происходит на стороне СУР

     \* @param[in] messages       сообщения

     \* @param[in] result                списки ниток, варианты результата работы алгоритма

     \* @return указатель на сериализованные данные

     \*/

     void deserializeWorkResult(const char\* data, std::string & messages, std::vector <std::vector <Nitka> > & result);

        /\*!

     \* десериализация  результата запуска алгоритмов для использования в многопоточном режиме

     \* происходит на стороне СУР

     \* @param[in] messages       сообщения

     \* @return указатель на сериализованные данные

     \*/

     void deserializeStartWorkResult(const char\* data, std::string & messages);

        /\*!

     \* десериализация результата работы алгоритмов

     \* происходит на стороне СУР

     \* @param[in] messages       сообщения

     \* @param[in] progress       прогресс выполнения задачи [0-1]

     \* @param[in] result                списки ниток, варианты результата работы алгоритма

     \* @return указатель на сериализованные данные

     \*/

     void deserializeGetResult(const char\* data, std::string & messages, double& progress, std::vector <std::vector <NitkaID> > & result);

        /\*!

     \* сохранение сериализованных данных в файл

     \* @param[in] filename       название файла

     \* @param[in] data           сериализованные данные

     \* @return успешно ли прошло сохранение

     \*/

        bool save(const char\* filename, const char\* data);

        /\*!

     \* загрузка сериализованных данных из файла

     \* @param[in] filename       название файла

     \* @param[in] data           сериализованные данные

     \* @return успешно ли прошла загрузка

    \*/

        bool load(const char\* filename, char\*& data);

        //! возвращает последнюю ошибку и очищает её

     std::string last\_error() { std::string res; std::swap(m\_last\_error, res); return res; }

        /\*

     \* запрос версии dll

     \* @return номер версии dll

     \*/

        static int version() { return s\_version; }

protected:

     /\*!

     \* чтение заголовка сериализованных данных

     \* @param[in] data  сериализованные данные

     \* @param[in] v версия протокола

     \* @return указатель на сериализованные данные с заголовком

     \*/

        const char\* writeHeader(const std::string & data, const CVersion& v);

        /\*!

     \* чтение заголовка сериализованных данных

     \* @param[in] data  сериализованные данные

     \* @param[in] v версия протокола

     \* @param[in] datasize размер данных в байтах

     \* @return указатель на сериализованные данные без заголовка

     \*/

        const char\* readHeader(const char\* data, const CVersion& v, uint64\_t& datasize);

        std::string m\_last\_error; //! последняя ошибка

        std::string m\_serialized\_data;  //! сериализованные данные

        static const int s\_version = 2; //! версия протокола

        //! начальные байты для соответствующих функций

     static const std::string s\_init\_magic;

     static const std::string s\_work\_magic;

        static const std::string s\_start\_work\_magic;

        static const std::string s\_init\_result\_magic;

        static const std::string s\_work\_result\_magic;

        static const std::string s\_start\_work\_result\_magic;

        static const std::string s\_get\_result\_magic;

};