**Входящая и исходящая очереди**

Необходимо сделать транспортную систему с помощью которой можно будет взаимодействовать с внешними системами: передавать и получать произвольные сообщения.

Общая концепция:

* входящая и исходящая очереди – разные объекты и таблицы
* очередь ничего не знает о сообщении, она просто транспорт
* для обработки сообщений реализуются экземпляры класса обработки сообщений, привязка экземпляров обработки идет к системам получателям/отправителям

Функционал очередей построен на следующих таблицах:

|  |  |  |
| --- | --- | --- |
| Таблица | Принадлежность | Описание |
| QUE\_OUT | Исходящая очередь | Непосредственно запись очереди с телом исходящего сообщения |
| QUE\_LOG | Исходящая очередь | логи отправки сообщения (может быть несколько на одну запись очереди) |
| QUE\_RecipientSystem | Исходящая очередь | Системы-получатели сообщений очереди, здесь указывается интерфейс-обработчик исходящего сообщения и его настройки |
| QUE\_EntriesForReview | Исходящая очередь | Записи в очередь на рассмотрении. Используется в случае формирования тела на основании функционала построителя JSON |
| QUE\_QRY\_RECIPT | Исходящая очередь | Связь экземпляра запроса и системы-получателя. Используется в случае формирования тела на основании функционала построителя JSON |
| QUE\_Handler\_tune | Исходящая и входящая очереди | Настройки обработчика очереди как для входящей, так и для исходящей очередей |
| QUE\_INCOME | Входящая очередь | Входящая очередь |
| QUE\_INCOME\_ANSWER | Входящая очередь | Возвращаемый ответ при вставке через API |
| QUE\_SenderSystem | Входящая очередь | Системы-отправитель сообщений очереди |

Структура таблиц и их взаимосвязь представлена ниже.

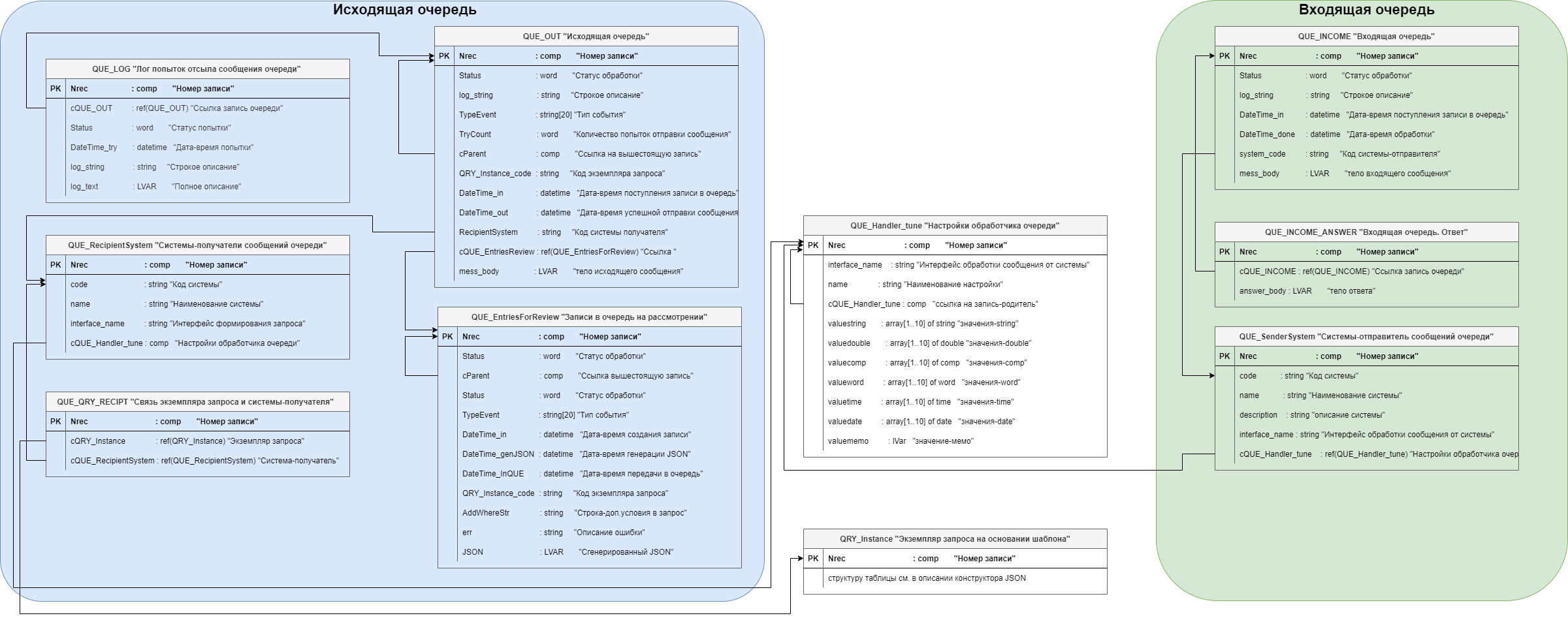


Рисунок 1. Структура и взаимосвязь таблиц очередей

В функционале очередей мы попытались реализовать следующий принцип: каждое отдельное действие обладает минимальным функционалом, чтобы оказывать минимальное воздействие на скорость работы всей системы в целом.

Исходя из этого принципа на текущий момент есть 2 пути добавления записи в очередь:

* непосредственное добавление записи в очередь;
* добавление записи в таблицу «QUE\_EntriesForReview», если тело будет формироваться на основе функционала конструктора JSON.

В случае непосредственного добавления тело сообщения формируется в момент работы пользователя с системой, и он будет ожидать окончания формирования сообщений и добавления его в очередь

В случае использования конструктора JSON нет необходимости формирования тела сообщения, в таблицу «QUE\_EntriesForReview» записывается лишь код экземпляра запросов и условие выборки (например, NREC записи). Сформирован JSON будет фоновым процессом, после чего добавлен в очередь. Таким образом задержки в работе программы будут минимальны.

Рассмотрим функционал записей на рассмотрении (работа с таблицей QUE\_EntriesForReview)

Данный функционал используется только совместно с функционалом конструктора JSON.

В функционале 3 основных шага, каждый из которых работает в отдельных фоновых задачах:

* добавление записей в таблицу QUE\_EntriesForReview (указывается код экземпляра запросов и условие выборки);
* генерация JSON по записям QUE\_EntriesForReview (используется функционал генерации JSON конструктора);
* перенос записей в исходящую очередь (в таблицу QUE\_OUT).

Логика работы всех трех шагов представлена ниже на схеме

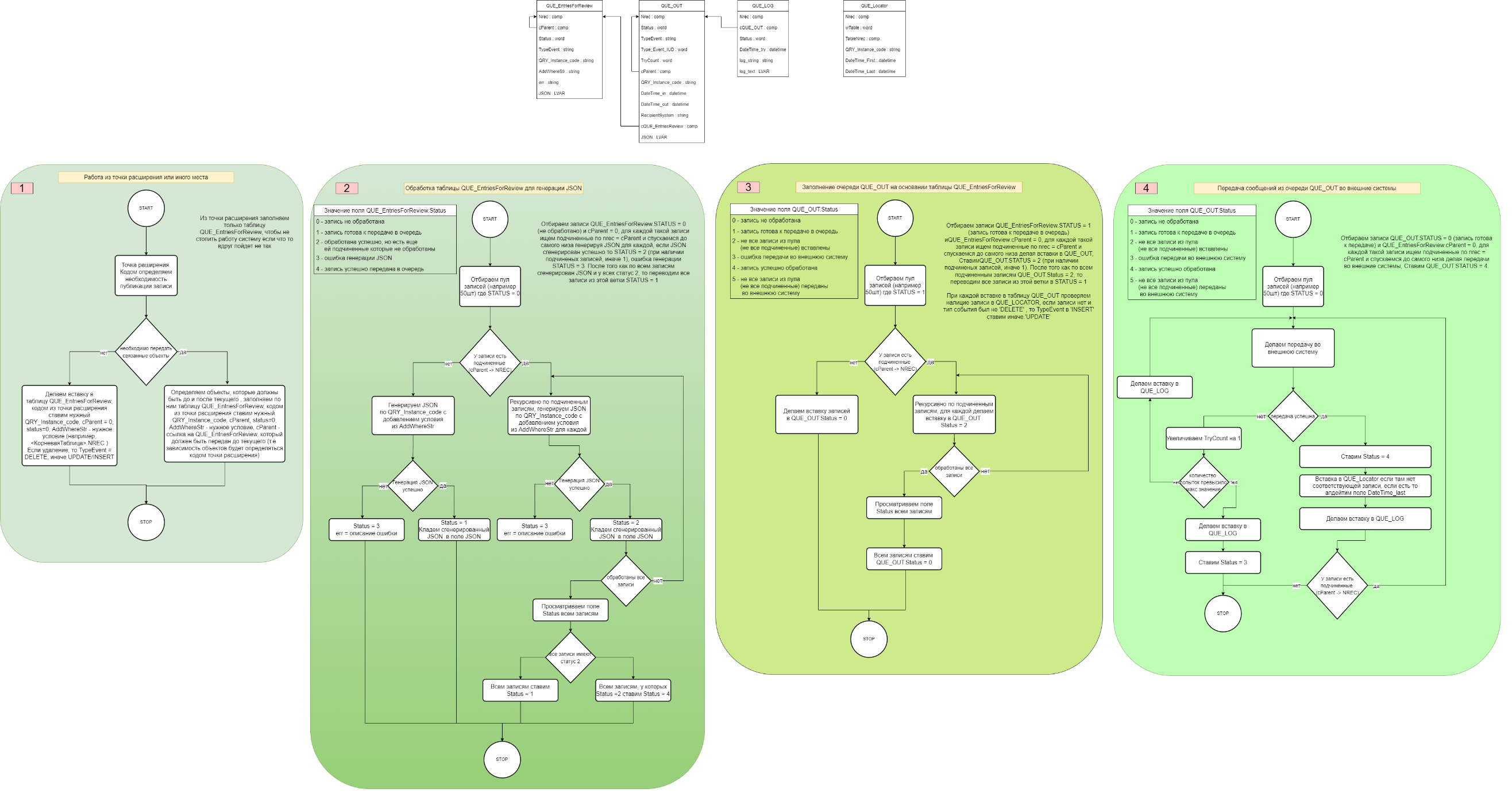


Рисунок. Общая схема функционала записей на рассмотрении

Данный функционал реализован классе QUE\_BASE. Ниже описание методов

#doc

Вставить запись из пула зависимых записей в очередь на рассмотрение для формирования JSON и последующей передаче в очередь

Возвращает NREC вставленной записи

У вставленной записи устанавливается статус "coPreQUE\_WaitForRelatedRcrdsToInsert = "ожидает вставки связанных записей" "

После того как все записи будут вставлены необходимо вызвать SetPoolRecordsReadyToGenerateJSON с указанием NREC самого верхнего элемента пула

так всем записям будет установлен статус "coPreQUE\_AwaitProcessJSONgenerate = "ожидает обработки для формирования JSON" "

и по ним будут генерироваться JSON процедурой ProcessEntriesForReview

\_cParent : comp - ссылка на вышестоящую запись. В случае пула зависимых записей

\_TypeEvent : string[20] - тип события coQUE\_TypeEvent\_Insert coQUE\_TypeEvent\_Delete coQUE\_TypeEvent\_Update

\_InstanceCode : string - код экземпляра запроса

\_AddWhereStr : string - строка для добавления в условие в экземпляр запроса

\_nrecRootTable : comp - nrec корневой таблицы запроса

если одновременно указано \_AddWhereStr и \_nrecRootTable, то берется \_nrecRootTable

Пример: сначала вставляем экземпляр запросов по генерации подразделений, получаем NREC вставленной записи,

далее вставляем все должности и всех сотрудников этого подразделения, указывая в качестве \_cParent для всех самый первый NREC

Для приказа с несколькими сотрудниками можно поступить так: вставляем первого сотрудника, под него второго, под него третьего и т.д.

под самого последнего вставляем сам приказ. Тем самым получаем последовательную выгрузку в очередь сначала всех сотрудников и только потом приказ по ним

#end

**Function** InsertEntriesForReview\_PoolParent(\_cParent : comp;

\_TypeEvent : string[20];

\_InstanceCode : string;

\_AddWhereStr : string = '';

\_nrecRootTable : comp= 0h

) : comp;

#doc

Вставить одиночную/независимую запись в очередь на рассмотрение для формирования JSON и последующей передаче в очередь

Возвращает NREC вставленной записи

У вставленной записи устанавливается статус "coPreQUE\_AwaitProcessJSONgenerate = "ожидает обработки для формирования JSON" "

т.е. по ним будут генерироваться JSON процедурой ProcessEntriesForReview

\_TypeEvent : string[20] - тип события coQUE\_TypeEvent\_Insert coQUE\_TypeEvent\_Delete coQUE\_TypeEvent\_Update

\_InstanceCode : string - код экземпляра запроса

\_AddWhereStr : string - строка для добавления в условие в экземпляр запроса

\_nrecRootTable : comp - nrec корневой таблицы запроса

если одновременно указано \_AddWhereStr и \_nrecRootTable, то берется \_nrecRootTable

#end

**Function** InsertEntriesForReview\_Single(\_TypeEvent : string[20];

\_InstanceCode : string;

\_AddWhereStr : string = '';

\_nrecRootTable : comp= 0h

) : comp;

#doc

Поставить в пуле записям в статусе "ожидает вставки связанных записей" статус в "ожидает обработки для формирования JSON"

апдейт начинается с нижних записей

\_cMainParent : comp - верхняя запись в пуле

#end

**Procedure** SetPoolRecordsReadyToGenerateJSON(\_cMainParent : comp);

#doc

Обработка записей в QUE\_EntriesForReview

Выборка пула записей, у которых cParent = 0h и статус = "coPreQUE\_AwaitProcessJSONgenerate "ожидает обработки для формирования JSON" "

Далее цикл по выбранным записям и рекурсивно генерируем по ним JSON функцией RecursiveGenerateJSON\_from\_EntriesForReviewRecords

Рекурсия нужна для обработки вложенных записей

#end

**procedure** ProcessEntriesForReview;

#doc

Обработка записей в QUE\_EntriesForReview

Выборка пула записей, у которых cParent = 0h и статус = coPreQUE\_ReadyToTransferToQueue "готова к передаче в очередь (JSON сформирован)" "

Далее цикл по выбранным записям и рекурсивно копируем в очередь

Рекурсия нужна для обработки вложенных записей

#end

**procedure** CopyEntriesForReviewToQueue;

Также возможно добавление сообщения непосредственно в очередь

Ниже описание данного метода (класс QUE\_BASE)

#doc

Положить сообщение в очередь.

Используется если сообщение сгенерировано не средствами конструктора JSON (компонент SNG\_QRY)

\_bodyMessage - tptr, ссылка на длинную строку с содержимым сообщения

\_codeReciptSystem - string, код системы-получателя (по ней определяется интерфейс для отсылки сообщения во внешнюю систему)

\_TypeEvent - тип события (coQUE\_TypeEvent\_Insert, coQUE\_TypeEvent\_Delete, coQUE\_TypeEvent\_Update)

#end

**function** InsertMessageDirectToQueue(\_bodyMessage : tptr; \_codeReciptSystem : string; \_TypeEvent : string) : boolean;

Далее запускается обработчик очереди.

Схема обработчика представлена ниже.

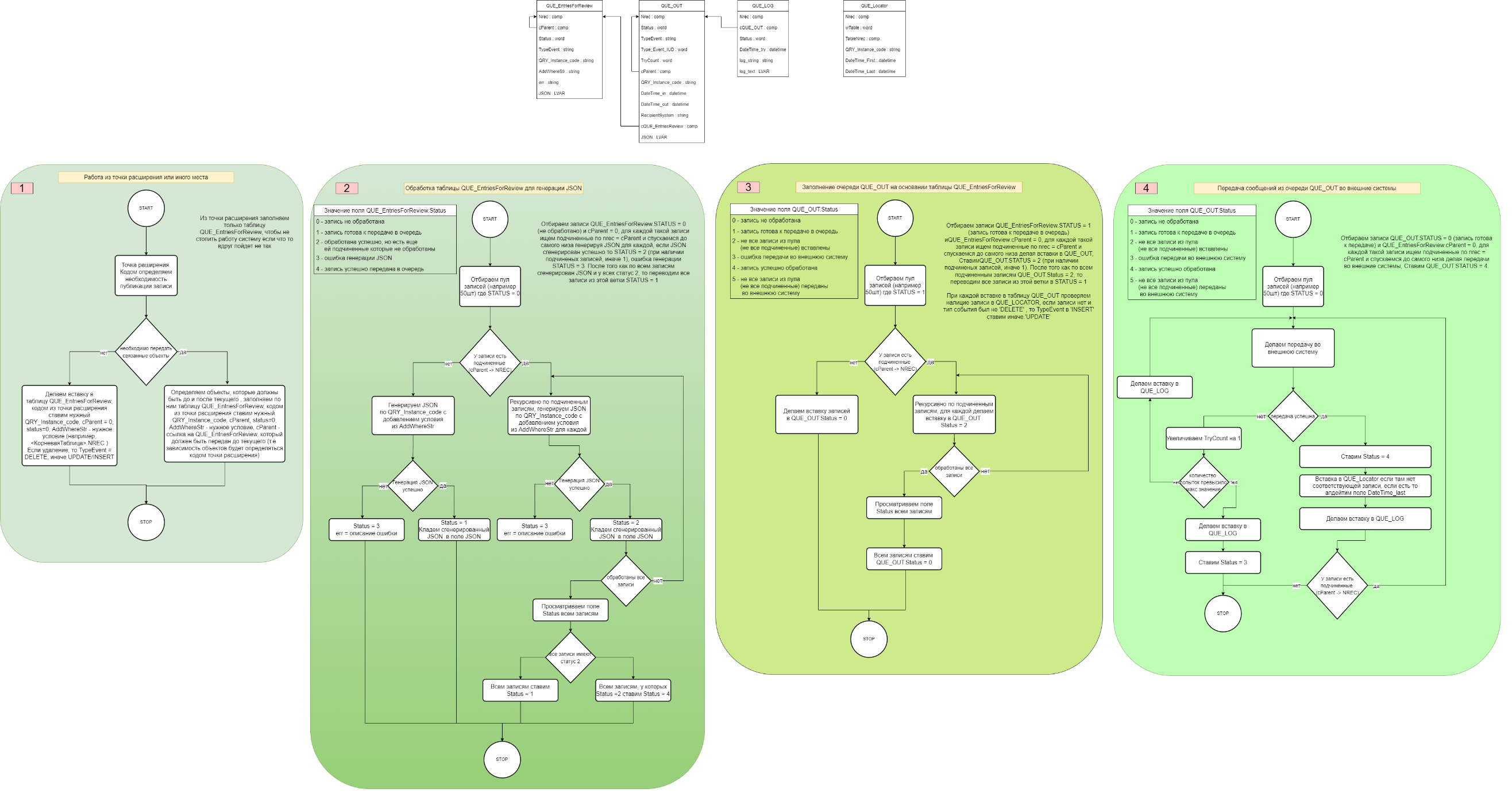


Рисунок. Схема обработки очереди.

Здесь в схеме есть таблица QUE\_LOCATOR. Сейчас ее нет в реализации.

Обработка очереди методом ProcessQueueOut.

#doc

Обработать записи очереди

#end

**procedure** ProcessQueueOut;

Обработка происходит по следующему принципу: по коду системы ищется запись в списке систем-получателей QUE\_OUT.RecipientSystem -> QUE\_RecipientSystem.code. Если запись не найдена, то в очередь записывается соответствующая ошибка.

Если же запись найдена, то вызывается интерфейс обработчик, который указан в системе-получателе QUE\_RecipientSystem.interface\_name с указанием настройки QUE\_RecipientSystem.cQUE\_Handler\_tune.

Интерфейс-обработчик записи очереди реализуется прикладным программистом как экземпляр объектного класса oQUE\_SendMessToExtSys.

Описание класса ниже.

**objinterface** oQUE\_SendMessToExtSys;

#doc

отправить сообщение во внешнюю систему

Возвращаемое значение:

TRUE - успешно

FALSE - неудачно

Параметры

\_cQueue - nrec QUE\_OUT

\_bodyMessage - указатель длинную строку, в которой содержится тело сообщения

\_InstanceCode - код экземпляра запроса

\_typeEvent - тип события (coQUE\_TypeEvent\_Insert, coQUE\_TypeEvent\_Delete или coQUE\_TypeEvent\_Update)

\_retStatus - возвращаемый, текстовое описание статуса (в случае ошибки краткое описание ошибки)

\_retFileLog - возвращаемый, путь на файл лога обработки/ошибки

#end

**function** SendMessage(\_cQueue : comp; \_bodyMessage : TpTr; \_InstanceCode: string; \_typeEvent: string; \_\_cQUE\_Handler\_tune : comp; var \_retStatus, \_retFileLog: string) : boolean;

#doc

Настройки интерфейса отправки сообщения во внешнюю систему

Интерфейс один, а настроек может быть много.

Настройка выбирается в интерфейсе регистрации систем получателей

\_\_cQUE\_Handler\_tune - ссылка на QUE\_Handler\_tune

#end

**procedure** Setup(\_\_cQUE\_Handler\_tune : comp);

#doc

Описание интерфейса

#end

**function** GetDescription : string;

По результатам работы обработчика пишется лог очереди и меняется статус сообщения.

Ниже скрины вышеописанного функционала.

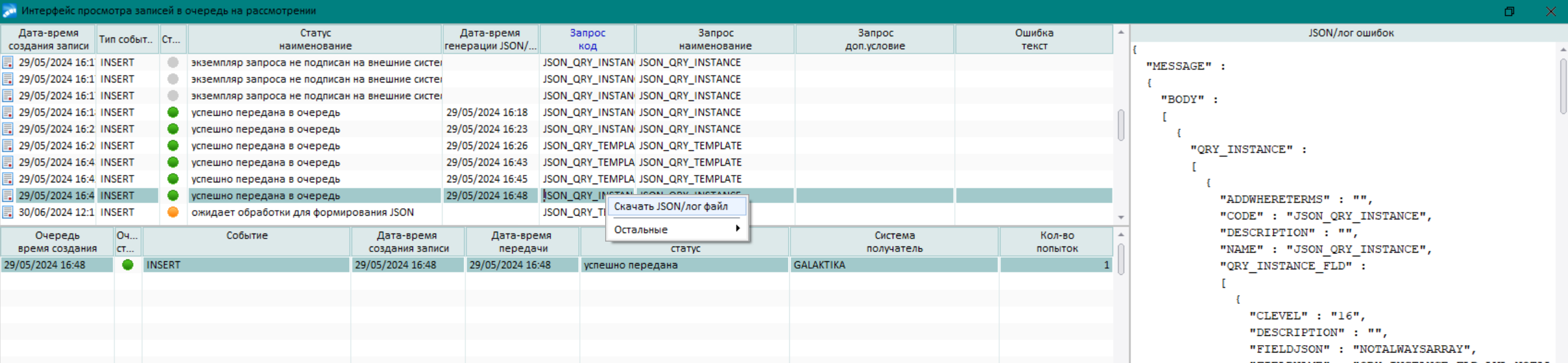


Рисунок. Интерфейс просмотра записей на рассмотрении

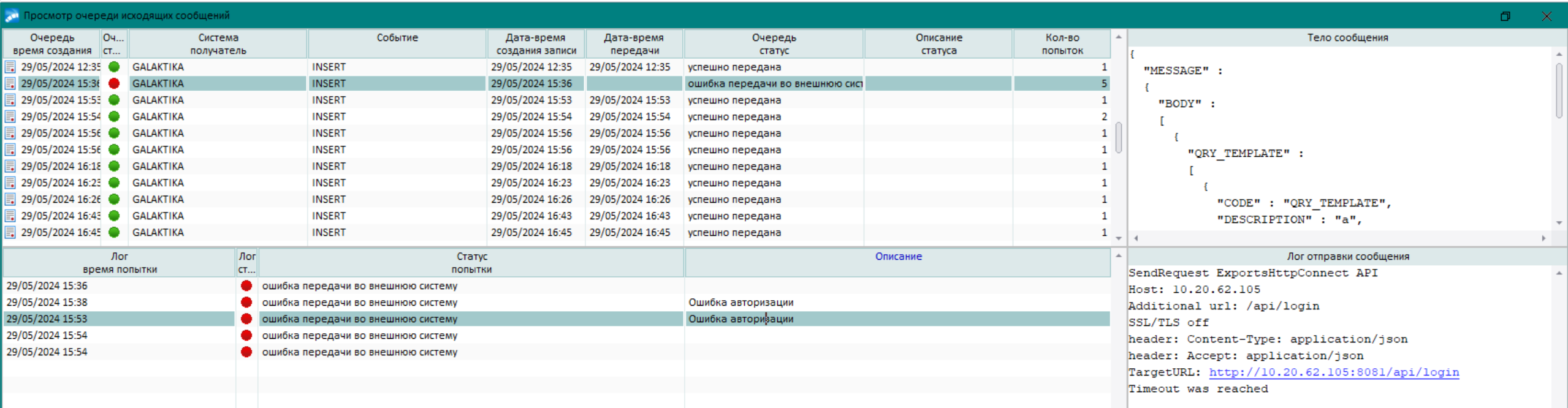


Рисунок. Интерфейс просмотра очереди исходящих сообщений

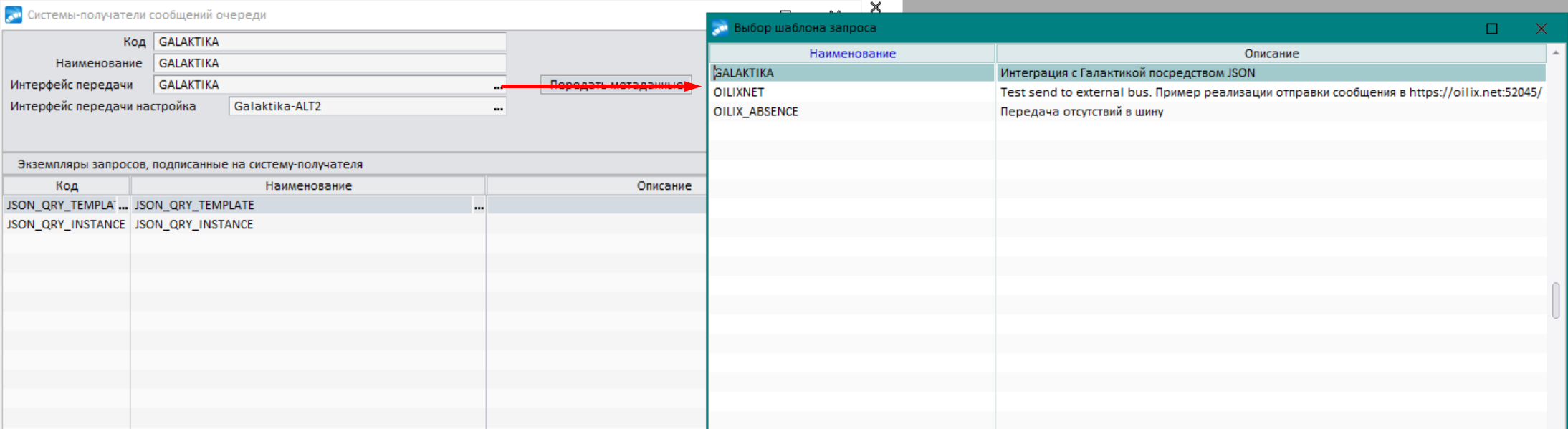


Рисунок. Интерфейс редактирования системы-получателя с выбором интерфейса передачи

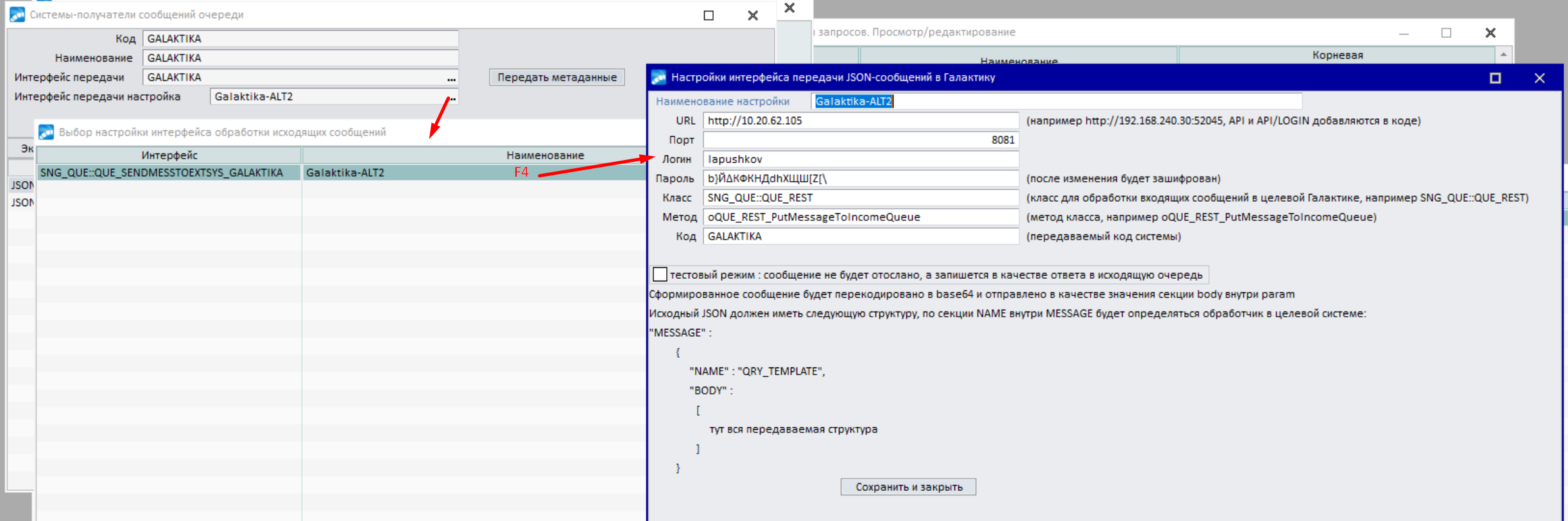


Рисунок. Интерфейс редактирования системы-получателя с выбором и редактированием настроек интерфейса передачи

Входящая очередь

Для приема сообщений реализован web класс oQUE\_REST

Описание методов дано ниже.

#doc

Интерфейс по работе с входящей очередью через REST

#end

**web** objInterface oQUE\_REST;

#doc

Положить сообщение во входящую очередь без обработки

Входные параметры (все STRING)

\_system\_code - код системы-отправителя

isbase64 - передаваемое тело в BASE64

\_mess - тело сообщения

ID\_QUEUE - возвращаемый. ID очереди

С учетом того, что входной формат это JSON, то корректно вложить \_mess (например если он тоже JSON или содержит спецсимволы)

в чистом виде не получится, поэтому рекомендую передавать \_mess как BASE64

Возвращаемый json

Code - код ответа (200 - успешно, 400 - Bad Request пустое тело, 401 - ошибка выполнения в Галактике)

Message - текстовое описание ответа

ID\_QUEUE - ID очереди для дальнейшего опроса статуса в GetStatusMessageInIncomeQueue

#end

**function** PutMessageToIncomeQueue(system\_code : string; isbase64 : boolean; body : widestring[2097151]) : TResultPutMessageToIncomeQueue;

#doc

Получить статус сообщения во входящей очереди

Входные параметры

ID\_QUEUE - string[36]

Возвращаемый json

status - наименование статуса

status\_code - код статуса (0 - готова в обработке, 1 - обрабатывается, 2 - ошибка, 3 - успешно, 4 - некорректный идентификатор очереди)

Body - тело ответа, содержимое определяется прикладным программистом

BodyBase64 - тело ответа, содержимое определяется прикладным программистом в BASE64

#end

**function** GetStatusMessageWithBody(ID\_QUEUE : string[36]): TResultGetStatusMessageInIncomeQueue;

Общая логика:

Внешняя программа по REST API вызывает метод PutMessageToIncomeQueue

Пример сообщения:

{

    "action":"atmcall"

  , "jsonrpc":"2.0"

  , "id":3

  , "class": "SNG\_QUE::QUE\_REST"

  , "method": "oQUE\_REST\_PutMessageToIncomeQueue"

  , "params":{"body": ""

            , "isbase64":**true**

            , "system\_code":"KATKAUSPKAU"

            }

}

Метод PutMessageToIncomeQueue перекодирует сообщение из BASE64 (если оно кодировано) и кладет его во входящую очередь, в ответ возвращается статус и ID очереди

Пример ответа:

{

    "reaction": "atmresult",

    "jsonrpc": "2.0",

    "result": {

        "Code": "200",

        "Message": "Запись успешно добавлена во входящую очередь",

        "ID\_QUEUE": "1407374883553546"

    },

    "id": 3

}

При необходимости можно запросить статус сообщения по его ID

Пример запроса статуса сообщения:

{

    "action":"atmcall"

  , "jsonrpc":"2.0"

  , "id":3

  , "class": "SNG\_QUE::QUE\_REST"

  , "method": "oQUE\_REST\_GetStatusMessageWithBody"

  , "params":{"ID\_QUEUE": "1407374883553546"

  }

}

Пример ответа по запросу статуса сообщения

{

    "reaction": "atmresult",

    "jsonrpc": "2.0",

    "result": {

        "status": "Готова к обработке",

        "status\_code": "0",

        "Body": "",

        "BodyBase64": ""

    },

    "id": 3

}

Далее обработчик входящей очереди обрабатывает сообщения и меняет их статус.

**ObjInterface** oQUE\_INCOME\_BASE;

#doc

Обработать записи очереди

#end

**procedure** ProcessQueueIncome;

Обработка сообщений входящей очереди аналогична исходящей: : по коду системы ищется запись в списке систем-отправителей QUE\_INCOME.system\_code -> QUE\_SenderSystem.code. Если запись не найдена, то в очередь записывается соответствующая ошибка.

Если же запись найдена, то вызывается интерфейс обработчик, который указан в системе-отправителе QUE\_SenderSystem.interface\_name с указанием настройки QUE\_SenderSystem.cQUE\_Handler\_tune.

Интерфейс-обработчик записи очереди реализуется прикладным программистом как экземпляр объектного класса oQUE\_ProcessMessIncomeQueue.

Описание класса ниже.

**objinterface** oQUE\_ProcessMessIncomeQueue;

#doc

Обработка входящего сообщения

Возвращаемое значение:

TRUE - успешно

FALSE - неудачно

Параметры

\_message - ссылка на длинную строку, содержащую сообщение. входной параметр

\_answer - ссылка на длинную строку-ответ. возвращаемый параметр

\_retMessage - текстовое описание ответа

#end

**function** ProcessMessage(\_message:TpTr; \_cQUE\_Handler\_tune : comp; var \_answerLong : tptr; var \_answer : string) : boolean;

#doc

Настройки интерфейса отправки сообщения во внешнюю систему

Интерфейс один, а настроек может быть много.

Настройка выбирается в интерфейсе регистрации систем получателей

\_\_cQUE\_Handler\_tune - ссылка на QUE\_Handler\_tune

#end

**procedure** Setup(\_\_cQUE\_Handler\_tune : comp);

#doc

Наименование интерфейса

#end

**function** GetDescription : string;

По результатам работы обработчика пишется лог очереди и меняется статус сообщения.

Ниже скрины вышеописанного функционала.

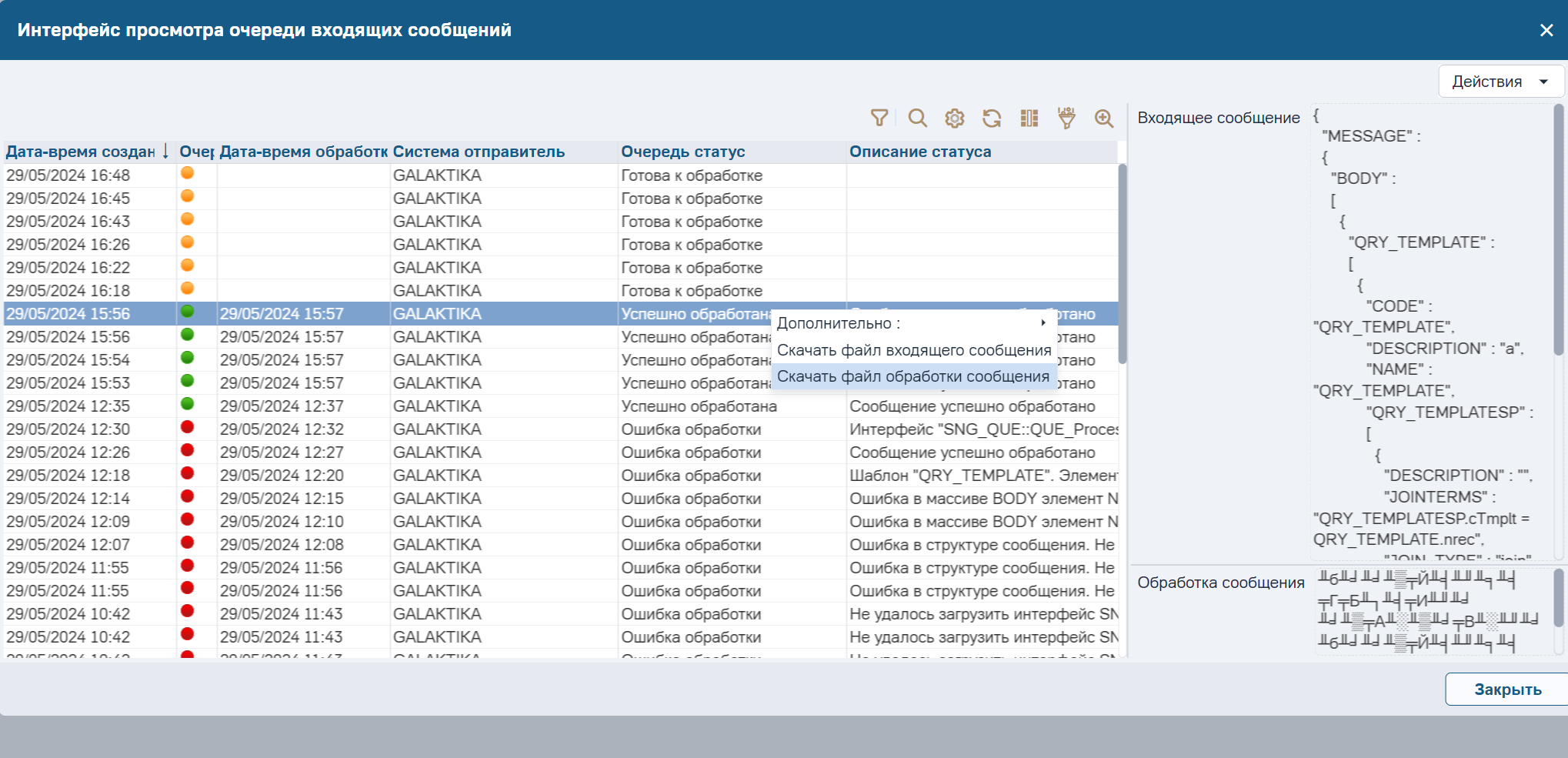


Рисунок. Интерфейс просмотра входящей очереди.

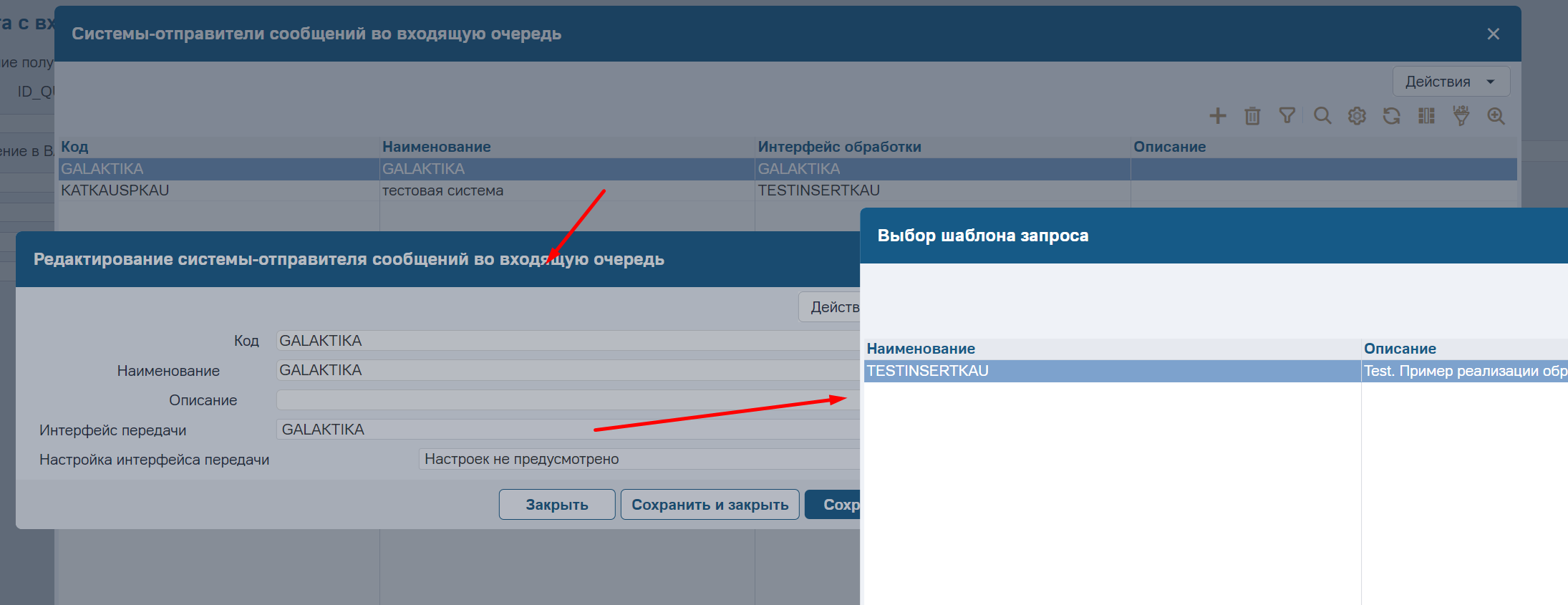


Рисунок. Интерфейс просмотра и редактирования систем-отправителей.