Справочники в Pyrus

Из Метабот можно передавать справочники в Pyrus. Для этого необходимо сделать интеграцию с PyrusApi.

Настройка таблиц

Для работы интеграции в боте необходимо создать таблицу pyrus_task_bindings:

ID	АКТИВНА	СИХР. СТРУКТУРЫ	ОТОБРАЖАТЬ В МЕНЮ	НАИМЕНОВАНИЕ	ЗАГОЛОВОК В ИНТЕРФЕЙСЕ	ДОСТУП ПО А
825	Да	Да	Да	pyrus_task_bindings	Привязка задач Pyrus	Нет

Структура таблицы:

АКТИВНО	СИХР. СТРУКТУРЫ	НАИМЕНОВАНИЕ	ЗАГОЛОВОК	ТИП	СВЯЗЫВАЕМАЯ ТАБЛИЦА	ОБЯЗАТЕЛЬНОЕ	ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ
Да	Да	id	ID	AUTOINCREMENT		Да	NULL
Да	Да	metabot_lead_id	ID лида	BIG_INT			NULL
Да	Да	pyrus_task_id	ID задачи	BIG_INT			NULL
Да	Да	channel_id	идентификатор канала	TEXT			
Да	Да	account_id	ID аккаунта интеграции	BIG_INT			NULL
Да	Да	is_active	Активен	TEXT			true
Да	Да	created_at	Дата создания	CREATED_AT		Да	NOW
Да	Да	updated_at	Дата обновления	UPDATED_AT			NULL
Да	Да	problem	Проблема	TEXT			

В плагине для интеграции будут использоваться поля pyrus_task_id и channel_id.

Настройка интеграции

Шаги подключения интеграции:

- Добавьте в ваш бизнес плагин Business.Integrations.PyrusApi со следующим кодом:

```
{
    _apiBase: 'https://api.pyrus.com/v4',
```

```
    _authUrl: 'https://accounts.pyrus.com/api/v4/auth',
    set apiBase(value) {
      this._apiBase = value
    },
    get apiBase() {
      return this._apiBase
    },
    set authUrl(value) {
      this._authUrl = value
    },
    get authUrl() {
      return this._authUrl
    },

    /**
     * HTTP-запрос к Pyrus API с Bearer-токеном.
     * @param {string} method - GET, POST, etc.
     * @param {string} uri - полный URI или путь относительно API_BASE
     * @param {Object|string|null} [body=null] - тело запроса (объект будет
JSON.stringify)
     * @returns {Object|null} - разобранный JSON ответ или null при ошибке
     */
    request: function(method, uri, body = null) {
      const url = uri.startsWith('http') ? uri : this._apiBase + (uri.startsWith('/') ?
uri : '/' + uri)
      const token = this.getToken()
      if (!token) return null
      const headers = {
        'Authorization': 'Bearer ' + token,
        'Content-Type': 'application/json'
      }

      try {

        api.request(method, url, [], [], null, body, headers)
        const code = api.getLastResponseCode()
        if (code === 200) {
          const content = api.getLastResponseContent()
          return content ? JSON.parse(content) : null
```

```
        }

        return false
      } catch (e) {
        debug(e.stack)
        return null
      }
    },

    /**
     * Получает токен из атрибутов бота (при необходимости можно вызвать refreshToken).
     * @returns {string|null}
     */
    getToken: function() {
      return bot.getAttr('pyrus_api_token') || null
    },

    /**
     * Обновляет токен через POST /api/v4/auth, сохраняет в bot.setAttr('pyrus_api_token',
...).
     * @returns {boolean} - успех
     */
    refreshToken: function() {
      const login = bot.getAttr('pyrus_api_login')
      const securityKey = bot.getAttr('pyrus_api_security_key')
      if (!login || !securityKey) {
        debug("Нет токенов")
        return false
      }
      const headers = { 'Content-Type': 'application/json' }
      try {
        api.request('POST', this._authUrl, [], [], null, { login: login, security_key:
securityKey }, headers)
        if (api.getLastResponseCode() !== 200) return false
        const content = api.getLastResponseContent()
        const data = content ? JSON.parse(content) : null
        const token = data?.access_token || data?.token
        if (token) {
          bot.setAttr('pyrus_api_token', token)
```

```
      return true
    }
    return false
  } catch (e) {

    debug(e.stack)
    return false
  }
},

/**
 * Ищет task_id в таблице pyrus_task_bindings по channel_id.
 * @param {string} channelId
 * @returns {number|null} - pyrus_task_id или null
 */
getTaskIdByChannelId: function(channelId) {
  if (!channelId) return null
  const rows = table.find('pyrus_task_bindings', [], [
    ['channel_id', '=', channelId]
  ])
  const first = rows?.[0]
  const id = first?.pyrus_task_id
  return id != null ? Number(id) : null
},

/**
 * Достаёт lead id из маппинга полей (поле MetabotLeadId).
 * @param {Array} mappings - массив { code, value }
 * @returns {number|null}
 */
getLeadIdByMapFields: function(mappings) {
  if (!Array.isArray(mappings)) return null
  const item = mappings.find(m => m?.code === 'MetabotLeadId')
  if (!item || item.value == null) return null
  const raw = String(item.value).split(',')[0].trim()
  const id = parseInt(raw, 10)
  return Number.isNaN(id) ? null : id
},
```

```
    /**
     * Определяет lead id по channel_id (формат "lead_<id>_dialog_...") или из маппинга.
     * @param {string} channelIdStr - строка channel_id
     * @param {Array} [mappingsFallback] - маппинг полей для fallback (getLeadIdByMapFields)
     * @returns {number|null}
     */
    getLeadIdByChannelId: function(channelIdStr, mappingsFallback) {
      if (channelIdStr && typeof channelIdStr === 'string') {
        const parts = channelIdStr.split('_')
        if (parts.length >= 2) {
          const id = parseInt(parts[1], 10)
          if (!Number.isNaN(id)) return id
        }
      }
      return this.getLeadIdByMapFields(mappingsFallback || [])
    },

    /**
     * GET /v4/tasks/{taskId}
     * @param {number} taskId
     * @returns {Object|null} - задача с form_id и т.д.
     */
    getTask: function(taskId) {
      return this.request('GET', '/tasks/' + taskId)
    },

    /**
     * GET /v4/forms/{formId}
     * @param {number} formId
     * @returns {Object|null} - форма с полями (fields)
     */
    getForm: function(formId) {
      return this.request('GET', '/forms/' + formId)
    },

    /**
     * GET /v4/catalogs/{catalogId}
     * @param {number} catalogId
```

```
     * @returns {Object|null} - справочник с items
     */
    getCatalog: function(catalogId) {
      return this.request('GET', '/catalogs/' + catalogId)
    },

    /**
     * Рекурсивный поиск поля по коду (info.code) или имени (name) в структуре формы.
     * @param {Array|Object} fields - массив полей или объект с полем fields
     * @param {string} fieldName - код поля (info.code) или название (name)
     * @returns {Object|null} - поле с id, name, type, info.catalog_id
     */
    getFieldByName: function(fields, fieldName) {
      if (!fields || !fieldName) return null
      const list = Array.isArray(fields) ? fields : (fields.fields || fields.items || [])
      for (let i = 0; i < list.length; i++) {
        const f = list[i]
        if (f?.info?.code === fieldName || f?.name === fieldName) return f
        if (f?.type === 'approval' && f?.steps) {
          const inSteps = this.getFieldByName(f.steps, fieldName)
          if (inSteps) return inSteps
        }
        if (f?.fields?.length) {
          const nested = this.getFieldByName(f.fields, fieldName)
          if (nested) return nested
        }
      }
      return null
    },

    /**
     * Ищет item_id в справочнике по значению в указанном столбце.
     * @param {number} catalogId
     * @param {string} value - искомое значение
     * @param {string} [columnName] - имя столбца для сравнения (например "Name",
"Наименование"). Если не задано, ищем по первому текстовому столбцу.
     * @returns {number|null} - item_id
     */
    getCatalogItemId: function(catalogId, value, columnName) {
```

```
      if (value == null || value === '') return null
      const catalog = this.getCatalog(catalogId)
      const items = catalog?.items || catalog?.catalog?.items
      if (!Array.isArray(items) || items.length === 0) return null
      const columns = catalog?.columns || catalog?.catalog?.columns || []
      const colName = columnName || (columns[0]?.name)
      const valueStr = String(value).trim().toLowerCase()
      for (let i = 0; i < items.length; i++) {
        const item = items[i]
        const itemId = item?.item_id ?? item?.id
        const cells = item?.cells || item?.values || []
        for (let c = 0; c < columns.length; c++) {
          const col = columns[c]
          if (colName && col?.name !== colName) continue
          const cell = cells[c] ?? cells[col?.id]
          const cellVal = (cell?.value != null ? cell.value : cell)
          if (cellVal != null && String(cellVal).trim().toLowerCase() === valueStr) {
            return itemId != null ? Number(itemId) : null
          }
        }
        if (!colName && cells?.length) {
          const firstVal = cells[0]?.value != null ? cells[0].value : cells[0]
          if (firstVal != null && String(firstVal).trim().toLowerCase() === valueStr) {
            return itemId != null ? Number(itemId) : null
          }
        }
      }
      return null
    },

    /**
     * POST /v4/tasks/{taskId}/comments с field_updates.
     * @param {number} taskId
     * @param {string} text - текст комментария
     * @param {Array} fieldUpdates - [{ id, value }, ...], value для справочника: { item_id: number }
     * @returns {Object|null}
     */
    updateTaskFields: function(taskId, text, fieldUpdates) {
```

```
      return this.request('POST', '/tasks/' + taskId + '/comments', {
        text: text || '',
        field_updates: Array.isArray(fieldUpdates) ? fieldUpdates : []
      })
    },

    /**
     * Строит массив field_updates по маппингу полей формы.
     * catalogMappings: [{ field_name|code, value, catalog_column? }, ...]
     * Для полей-справочников ищет item_id по value в каталоге, для остальных подставляет value как есть.
     * @param {Object|number} formOrFormId - объект формы с полем fields или id формы
     * @param {Array} catalogMappings
     * @returns {Array} - field_updates для updateTaskFields
     */
    buildFieldUpdates: function(formOrFormId, catalogMappings) {
      if (!Array.isArray(catalogMappings) || !catalogMappings.length) return []
      const isFormObject = formOrFormId != null && typeof formOrFormId === 'object' && Array.isArray(formOrFormId.fields)
      const formFields = isFormObject ? formOrFormId.fields : (() => {
        const form = this.getForm(formOrFormId)
        return form?.fields || form?.form?.fields || []
      })()
      const updates = []
      for (const m of catalogMappings) {
        const fieldName = m?.field_name || m?.code
        const value = m?.value
        if (fieldName == null || value == null) continue
        const field = this.getFieldByName(formFields, fieldName)
        if (!field?.id) continue
        const fieldType = (field.type || '').toLowerCase()
        const catalogId = field.info?.catalog_id ?? field.catalog_id
        if (catalogId && ['catalog', 'choice', 'lookup'].includes(fieldType)) {
          const col = m?.catalog_column
          if (Array.isArray(value)) {
            const itemIds = value
              .map(v => this.getCatalogItemId(catalogId, v, col))
              .filter(id => id != null)
```

```
            if (itemIds.length) updates.push({ id: field.id, value: { item_ids: itemIds }
})
          } else {
            const itemId = this.getCatalogItemId(catalogId, value, col)
            if (itemId != null) updates.push({ id: field.id, value: { item_id: itemId } })
          }
        } else {
          updates.push({ id: field.id, value: value })
        }
      }
      return updates
    }
  }
```

- В Атрибуты бота добавьте следующие системные переменные:
  - **pyrus_api_login** - логин аккаунта Pyrus
  - **pyrus_api_security_key** - ключ доступа к АПИ
  - **pyrus_api_token** - токен авторизации АПИ
  - **refreshTokenTime** - время для обновления pyrus_api_token (по-умолчанию поставьте 0)

ТИП	НАЗВАНИЕ	ЗН
variable **[системный]**	pyrus_api_login	st
variable **[системный]**	pyrus_api_security_key	XA
variable **[системный]**	pyrus_api_token	ey Nl hD N: C. wi N: Sj
variable **[системный]**	refreshTokenTime	17

- В место где хотите передавать справочник добавьте код обновления заявки:

```
let pyrusApi = snippet("Business.Integrations.PyrusApi")
let task_id = pyrusApi.getTaskIdByChannelId(channelId)

let mappings = [{
               "code": "CustomCatalog1",
               "value": [lead.getAttr('Подотдел'), lead.getAttr('Структура')]
           }]

let now = new Date();
let refreshTokenTime = now.getTime() + (1000*60*60*24);

if (task_id) {
  if(bot.getIntAttr('refreshTokenTime') <= now.getTime()){
    bot.setAttr('refreshTokenTime', refreshTokenTime)
    pyrusApi.refreshToken()
  }
  let task = pyrusApi.getTask(task_id)
  let formId = task?.task?.form_id

  if (formId) {
    let form = pyrusApi.getForm(formId)
    // Получаем под капотом все данные полей и самостоятельно маппим
    let fieldUpdates = pyrusApi.buildFieldUpdates(form, mappings)
    if (fieldUpdates?.length) {
      pyrusApi.updateTaskFields(task_id, "Поля формы обновлены", fieldUpdates)
    }
  }
}
```

Разберем код:

- В первой строке обращаемся к плагину PyrusApi.
- Во второй забираем из таблицы заявок id заявки по id канала.
- Объявляем передаваемые параметры в **mappings**. Поле **code** должно соответствовать коду поля справочника в форме в Pyrus, а **value** данным из справочника.

Второй способ передачи справочников:

```
{
  "code": "Тарифы", - По названию
  "value": "Стартовый" - Единичный выбор
}
```

- Далее вычисляем пришло ли время обновления токена, если да, то обновляем и записываем новые данные в атрибуты бота.
- В 17 строке находим задачу в Pyrus через АПИ.
- В 21 находим форму в Pyrus через АПИ.
- В 23-25 обновляем поля задачи.

> Данный функционал работает для передачи любых полей формы

Версия #2
Ирина Петрова создал 3 April 2026 11:10:33
Ирина Петрова обновил 7 April 2026 08:17:06