MorphCluster Registry

MorphCluster Registry — это пакет, расширяющий MorphCluster Core и предоставляющий сервисы для централизованного хранения и управления настройками (конфигурацией) системы. Он включает два сервиса:

- **CommonRegistry** — базовое персистентное хранилище настроек в виде JSON-файла с возможностью расширения схемы данных.
- **RegistryDummy** — адаптер, обеспечивающий обратную совместимость и предоставляющий старый интерфейс реестра, делегируя вызовы в CommonRegistry.

Оба сервиса строятся на основе `ServiceRequire` из ядра MorphCluster Core и могут работать как локально, так и в составе распределённой системы через глобальные сервисы.

Установка и подключение

```
import { CommonRegistry, RegistryDummy } from '@morphcluster/registry';
```

Пакет автоматически регистрирует схемы сервисов и готов к использованию в `ServiceHost`.

CommonRegistry

`CommonRegistry` — это сервис, реализующий персистентное хранилище произвольных JSON-данных с возможностью валидации и расширения схемы. Данные сохраняются в файл `common-registry-data.json`, а схема — в `common-registry-schema.json` внутри директории данных приложения (`Config.dataDir`).

Наследуется от `ServiceRequire`, поэтому может ожидать запуска других сервисов перед собственным стартом (по умолчанию зависимостей нет).

Инициализация

```
const registry = new CommonRegistry(host);
// host — экземпляр ServiceHost
```

В конструкторе:

- инициализируется схема сервиса через `setSvcSchema(serviceSchema)` (импортируется из `./service-schema.js`);
- создаётся стартовая JSON-схема реестра (`registrySchema`) на основе `defaultSchema` — объект с единственным возможным ключом `main`, содержащим поля `name` и

`description` (оба необязательные);
- создаются события `onChanged` и `onSchemaRequest`.

Свойства

Свойство	Тип	Описание
`data`	`object`	Текущие данные реестра. Изначально пустой объект, заполняется при старте из файла.
`registrySchema`	`object`	Текущая JSON-схема, описывающая допустимую структуру данных. Может расширяться вызовом `mergeSchema`.
`onChanged`	`Event`	Срабатывает при любом изменении данных (`set` или `merge`). В событии передаётся рабочее пространство `"common"`.
`onSchemaRequest`	`Event`	Событие запроса на отправку схемы (может использоваться для синхронизации).

Методы запросов (Request Handlers)

Все методы требуют прав администратора (`needAdmin: true`), за исключением отсутствующих явных проверок в коде — согласно схеме, все запросы помечены `needAdmin: true`. Доступ к ним осуществляется через стандартный механизм вызова сервиса (`sendRequest`).

`get()`

Возвращает текущие данные реестра.

```
const data = await registry.sendRequest('get', {}, log);
// data = { ... }
```

`set({ data })`

Полностью заменяет данные реестра новым объектом. Автоматически сохраняет изменения на диск и генерирует событие `onChanged`.

```
await registry.sendRequest('set', {
  data: {
    main: { name: 'MyServer', description: 'Основной сервер' },
    features: { darkMode: true }
  }
}, log);
```

merge({ data })

Глубоко объединяет переданный объект с текущими данными (используется `deepmerge`). Изменения сразу пишутся на диск и вызывают `onChanged`.

```
await registry.sendRequest('merge', {
  data: { features: { newUI: false } }
}, log);
```

mergeSchema({ schema })

Расширяет JSON-схему реестра путём рекурсивного наложения (`JsonSchema.overlay`). Новая схема записывается в файл.

```
await registry.sendRequest('mergeSchema', {
  schema: {
    properties: {
      features: {
        type: 'object',
        properties: {
          darkMode: { type: 'boolean' },
          newUI: { type: 'boolean' }
        }
      }
    }
  }
}, log);
```

getSchema()

Возвращает текущую полную JSON-схему реестра.

```
const schema = await registry.sendRequest('getSchema', {}, log);
```

clearSchema()

Сбрасывает схему реестра до значения по умолчанию (`defaultSchema`) и сохраняет её.

```
await registry.sendRequest('clearSchema', {}, log);
```

Жизненный цикл

При старте сервиса (`start(log)`):

1. Определяются пути `dataDir`, `registrySchemaPath`, `dataPath`.
2. Вызывается `super.start(log)` для ожидания обязательных зависимостей.
3. Выполняется попытка загрузить схему из файла `common-registry-schema.json`. Если файл отсутствует или повреждён, в лог пишется ошибка уровня 50.
4. Аналогично загружаются данные из `common-registry-data.json`. При отсутствии файла `this.data` остаётся пустым объектом, ошибка логируется.

Сервис не переопределяет `stop()` (используется стандартный из `ServiceRequire`).

RegistryDummy

`RegistryDummy` — сервис, предоставляющий интерфейс реестра, совместимый с ранними версиями MorphCluster. Он оборачивает `CommonRegistry`, добавляя поддержку «рабочих пространств» и метод `delete`.

Наследуется от `ServiceRequire` и требует `CommonRegistry` в качестве обязательной зависимости.

Инициализация

```
const dummy = new RegistryDummy(host);
```

В конструкторе:

- регистрируется схема сервиса `Registry` (из `./service-schema.js`);
- объявляется требование `CommonRegistry`;
- создаются события `onChanged` и `onSchemaRequest`.

Свойства

Свойство	Тип	Описание
`CommonRegistry`	`CommonRegistry`	Ссылка на экземпляр `CommonRegistry`, заполняется после `super.start()`.
`onChanged`	`Event`	Проксирует событие `onChanged` от CommonRegistry, но с workspace `"main"`.
`onSchemaRequest`	`Event`	Аналогично событию из CommonRegistry (может использоваться для уведомлений о необходимости отправить схему).

Методы запросов

`listWorkspaces()`

Возвращает статический список рабочих пространств. В текущей реализации — только одно пространство `"main"`.

```
const { workspaces } = await dummy.sendRequest('listWorkspaces', {}, log);
// workspaces = ["main"]
```

getAll()

Возвращает данные всех рабочих пространств. Для пространства `"main"` берутся текущие данные из `CommonRegistry.data`, к ним добавляются поля `name: 'main'` и `dummy: true`.

```
const { workspaces } = await dummy.sendRequest('getAll', {}, log);
// workspaces: [{ name: 'main', dummy: true, ...data }]
```

delete({ name })

В `RegistryDummy` не реализован — всегда выбрасывает ошибку `"Not available in RegistryDummy"`.

set({ project })

Заменяет данные реестра, вызывая `CommonRegistry.set({ data: project })`.

```
await dummy.sendRequest('set', { project: { ... } }, log);
```

merge({ project })

Дополняет данные через `CommonRegistry.merge({ data: project })`.

mergeSchema({ schema })

Проксирует вызов `CommonRegistry.mergeSchema`.

getSchema()

Проксирует вызов `CommonRegistry.getSchema`.

clearSchema()

Проксирует вызов `CommonRegistry.clearSchema`.

Жизненный цикл

- **start(log)**: вызывает `super.start(log)` для получения доступа к `CommonRegistry`, затем подписывается на событие `CommonRegistry.onChanged` и при его срабатывании генерирует собственное `onChanged` с workspace `"main"`.
- **stop()**: вызывает `super.stop()`, затем отписывается от `CommonRegistry.onChanged`.

Схемы сервисов

CommonRegistry

Запрос	Параметры	Ответ	Описание
`get`	—	`object`	Получить все настройки
`set`	`data: object`	—	Заменить настройки целиком
`merge`	`data: object`	—	Глубоко дополнить настройки
`mergeSchema`	`schema: object`	—	Расширить JSON-схему реестра
`getSchema`	—	`object`	Получить текущую схему реестра
`clearSchema`	—	—	Сбросить схему до стандартной

События:

- `onChanged` — настройки изменились (параметр workspace = `"common"`);
- `onSchemaRequest` — запрос на отправку схем реестра.

Registry (RegistryDummy)

Запрос	Параметры	Ответ	Описание
`listWorkspaces`	—	`{ workspaces: string[] }`	Список пространств (всегда `["main"]`)
`getAll`	—	`{ workspaces: object[] }`	Данные всех пространств
`delete`	`name: string`	—	**Не поддерживается** (выбрасывает ошибку)
`set`	`project: object`	—	Замена настроек пространства `main`
`merge`	`project: object`	—	Дополнение настроек
`mergeSchema`	`schema: object`	—	Расширение схемы реестра
`getSchema`	—	`object`	Текущая схема реестра
`clearSchema`	—	—	Сброс схемы

События:

- `onChanged` — настройки изменились (параметр workspace = `"main"`);

- `onSchemaRequest` — запрос схем реестра.

Пример использования

```
import { ServiceHost, Config, Logger, LoggerBackendConsole } from '@morphcluster/core';
import { CommonRegistry, RegistryDummy } from '@morphcluster/registry';

const config = new Config({ /* ... */ });
await config.load();

const host = new ServiceHost('main');
host.Config = config;
host.Logger = new Logger({ host: 'main' });
host.Logger.backends = [new LoggerBackendConsole()];

// Добавляем базовый реестр
const commonReg = new CommonRegistry(host);
host.addService(commonReg, 'CommonRegistry');

// Добавляем адаптер совместимости
const dummyReg = new RegistryDummy(host);
host.addService(dummyReg, 'Registry');

await host.start(host.Logger);

// Используем адаптер для установки настроек
await dummyReg.sendRequest('set', {
  project: {
    main: { name: 'Prod', description: 'Production server' }
  }
}, host.Logger.createWriter('Initial config'));

// Читаем через CommonRegistry
const data = await commonReg.sendRequest('get', {}, host.Logger.createWriter('Read config'));
console.log(data);
```

Интеграция с другими сервисами

Оба сервиса могут быть зарегистрированы в `GlobalServices` и использоваться удалённо через `ServiceBridge` или напрямую через `GlobalService`. Благодаря наследованию от `ServiceRequire`, они легко встраиваются в цепочки зависимостей. Например, сервис аутентификации может требовать `CommonRegistry` для хранения настроек политик безопасности.

```
class MyAuthService extends ServiceRequire {
  constructor(host) {
    super(host);
    this.requirements = ['CommonRegistry'];
  }

  async start(log) {
    await super.start(log);
    const cfg = await this.CommonRegistry.sendRequest('get', {}, log);
    // использовать cfg.policies...
  }
}
```

Примечания

- Данные реестра хранятся в файлах без шифрования. Для безопасности критичных данных рекомендуется ограничивать права доступа к директории `dataDir`.
- Схема реестра (`registrySchema`) используется для валидации через `JsonSchema.sanitize` или другие инструменты MorphCluster Core.
- `RegistryDummy` существует для плавного перехода со старого API. В новых проектах рекомендуется использовать `CommonRegistry` напрямую.
- При отсутствии файлов схемы и данных сервис стартует с пустыми значениями, записывая ошибки в лог (уровень 50). Это позволяет системе не падать при первом запуске.

Revision #2
Created 21 June 2026 15:42:35 by Георгий
Updated 26 June 2026 13:52:59 by Георгий