Adding event handler step by step

Default settings

The default event handler of the Sync module is AtomFeed. You can find more information about it on its wiki page.

Adding event configuration mappings

The Sync module reads event basing on defined configuration.

To add a new event handler, you have to map required settings to EventConfiguration by implementing EventConfigurationMapper interface. You can check AtomFeed implementation as an example:

Source

AtomfeedEventConfigurationMapperImpl

```
@Component("sync2.eventConfigurationMapper." + SyncConstants.ATOMFEED_EVENT_HANDLER)
public class AtomfeedEventConfigurationMapperImpl implements EventConfigurationMapper<FeedConfiguration> {

        @Override
        public EventConfiguration map(FeedConfiguration feedConfiguration) {
                return new EventConfiguration(feedConfiguration.getLinkTemplates());
        }
}
```

You will also have to implement EventConfigurationService interface. You can check AtomFeed implementation as an example:

Source

AtomfeedEventConfigurationServiceImpl

```
@Component("sync2.eventConfigurationService." + SyncConstants.ATOMFEED_EVENT_HANDLER)

public class AtomfeedEventConfigurationServiceImpl implements EventConfigurationService {

        @Autowired
        private FeedConfigurationService feedConfigurationService;

        @Autowired
        @Qualifier("sync2.eventConfigurationMapper.atomfeed")
        private EventConfigurationMapper<FeedConfiguration> eventConfigurationMapper;

        @Override
        public EventConfiguration getEventConfigurationByCategory(CategoryEnum categoryEnum) {
                FeedConfiguration feedConfiguration = feedConfigurationService
                                .getFeedConfigurationByCategory(categoryEnum.getCategory());
                return eventConfigurationMapper.map(feedConfiguration);
        }
}
```

Implementing feed readers

In order to create new feed readers you have to implement ParentFeedReader and LocalFeedReader interfaces.

LocalFeedReader interface is responsible for handling event processing as a child instance and an example can be found here:

LocalAtomfeedFeedReaderImpl

```
@Component("sync2.localFeedReader." + SyncConstants.ATOMFEED_EVENT_HANDLER)
public class LocalAtomfeedFeedReaderImpl extends AbstractAtomfeedFeedReader implements LocalFeedReader {

        public LocalAtomfeedFeedReaderImpl() {
                super(new LocalAtomfeedFeedWorker());
        }

        @Override
        protected SyncMethodConfiguration getSyncMethodConf() {
                return configurationService.getSyncConfiguration().getPush();
        }

        @Override
        protected String getBaseUri() {
                return configurationService.getSyncConfiguration().getGeneral().getLocalFeedLocation();
        }

        @Override
        public void readAndPushAllFeeds() {
                readAndProcessAllFeeds();
        }

        @Override
        public void readAndPushAllFeeds(CategoryEnum category) throws SyncException {
                readAndProcessFeedsForCategory(category.getCategory());
        }
}
```

ParentFeedReader interface is responsible for handling event processing as a parent instance and an example can be found here:

ParentAtomfeedFeedReaderImpl

```
@Service("sync2.parentFeedReader." + SyncConstants.ATOMFEED_EVENT_HANDLER)
public class ParentAtomfeedFeedReaderImpl extends AbstractAtomfeedFeedReader implements ParentFeedReader {

        public ParentAtomfeedFeedReaderImpl() {
                super(new ParentAtomfeedFeedWorker());
        }

        @Override
        protected SyncMethodConfiguration getSyncMethodConf() {
                return configurationService.getSyncConfiguration().getPull();
        }

        @Override
        protected String getBaseUri() {
                return configurationService.getSyncConfiguration().getGeneral().getParentFeedLocation();
        }

        @Override
        public void pullAndProcessAllFeeds() {
                readAndProcessAllFeeds();
        }

        @Override
        public void pullAndProcessFeeds(CategoryEnum category) throws SyncException {
                readAndProcessFeedByCategory(category.getCategory());
        }
}
```

Naming convention of event handler related beans

All the Spring beans created in steps above have specific bean name. It is important to change only the last part of this name. The Sync 2.0 module chooses the event handler based on specific prefix and global property ("sync2.event.handler"). For instance the AtomfeedEventConfigurationServiceImpl bean has name 'sync2.eventConfigurationService.atomfeed' so the constant prefix it is the 'sync2.eventConfigurationService.' and the 'atomfeed' is the suffix. You can found an implementation of that convention here.

Setting (Formerly Global property from platform 1.9 upwards)

In order to use newly created event handler you have to specify it in setting (formerly global properties from platform 1.9 upwards). Learn more about settings here

To do this, run your OpenMRS instance. From Home page go to System Administration > Advanced Administration > Maintenence > Advanced Settings and set value of this property:

Setting (Formerly Global property from platform 1.9 upwrads) name	Default value
sync2.event.handler	atomfeed

Note! As you can see in the table above the default event handler is AtomFeed. Please make sure that a property value of the new event handler will be the same as the suffix of every Sync2 bean related to this event handler. For example the ParentAtomfeedFeedReaderImpl bean is qualified by 'sync2.parentFeedReader.atomfeed'. In that case the sync2.event.handler property must be equal to 'atomfeed'.