The Future of Rails I18n

http://rails-i18n.org

The future of Rails I18n

Official Hymne: Painkiller (Judas Priest)

The future of Rails I18n

Official Hymne: Painkiller (Judas Priest)

Planets devastated
Mankinds on its knees
A saviour comes from out the skies
In answer to their pleas

This is the painkiller

With mankind ressurrected
Forever to survive
Returns from armageddon to the skies

The future of Rails I18n

Sven Fuchs & Marko Seppä

Welcome everybody!

I'm very happy to talk about "The future of Internationalization in Ruby on Rails" today.

About me

This never occurred to me before ...

So, who’s me?

This never occurred to me before ...

But in a recent RailsEnvy podcast I’ve learned that my name could be pronounced like this ...

Sven F*chs

... at least that's what RailsEnvy says ;-)

... errr, but ...

Sven Fuchs

for better results in Google use this

if you want to google me you probably have better luck with this.

Sven Fuchs

- Berlin

I'm currently living in Berlin and I love it :)

Sven Fuchs

- Berlin
- Learned Assembler on C64 in ~ 1984

I started programming in 1984

Sven Fuchs

- Berlin
- Learned Assembler on C64 in ~ 1984
- (Web) Developer since ~ 1996

I've been in business as a developer, mostly as a web developer for some years and ...

Sven Fuchs

- Berlin
- Learned Assembler on C64 in ~ 1984
- (Web) Developer since ~ 1996
- 99% Ruby + Rails since ~ 3 years

... switched to working with Ruby and then Ruby on Rails a couple of years ago.

Sven Fuchs

- Led Rails I18n group

Some of you might know me because I've **led the Rails I18n group** for the last couple of months and ...

Sven Fuchs

- Led Rails I18n group
- Implemented Rails I18n API patch

... did the **final touches** on the implementation of the new **API** in Rails recently.

Sven Fuchs

- Led Rails I18n group
- Implemented Rails I18n API patch
- Globalize tutorial series

Some people know my **blog** because of some **tutorials** I've written **about Globalize**.

Sven Fuchs

- Led Rails I18n group
- Implemented Rails I18n API patch
- Globalize tutorial series
- Currently working on adva-cms

... and I'm currently working on a CMS application platform called adva_cms.

Of course, as a **German developer** I've **always been interested** in support for **Internationalization**

so I'm **really happy that I've able to contribute** something in this area to Ruby on Rails.

Questions?

If there are any questions ...

Questions

At the end of the talk, please.

please try to **make a mental note** and keep it for the end.

I've **set some time** aside for Q&A at the end of the talk

The future of Rails I18n

Ok, by the end of this talk I hope you'll have a clear picture of

The future of Rails I18n

- Overview: the Rails I18n project

What the Rails Internationalization project is ...

The future of Rails I18n

- Overview: the Rails I18n project
- Rails I18n: what's in the box?

What’s included in the new Rails Internationalization API

And what’s not

The future of Rails I18n

- Overview: the Rails I18n project
- Rails I18n: what's in the box?
- How to use it

How you can use it.

The future of Rails I18n

- Overview: the Rails I18n project
- Rails I18n: what's in the box?
- How to use it (brief demo)

I will give a very brief demo

Of course this is a huge topic

so I'll point you to where you can find more resources.

The future of Rails I18n

- Overview: the Rails I18n project
- Rails I18n: what's in the box?
- How to use it (brief demo)
- Joshua Harvey about Globalize 2

Then we have a short and sweet Interview video prepared with Joshua Harvey

the original author of Globalize

The future of Rails I18n

- Overview: the Rails I18n project
- Rails I18n: what's in the box?
- How to use it (brief demo)
- Joshua Harvey about Globalize 2
- How you can contribute

And finally I'll give you some hints about

what we're going to do in the future

and how you can contribute to that

So

To better understand the future of Internationalization in Rails ...

The history of Rails I18n

I want to contrast that a bit with the **history** that we all suffered through.

As you know the history of Internationalization in Rails is **really, really dark**.

Chaos

These were times of chaos.

Chaos

- Unclear

They were unclear

Chaos

- Unclear
- Unpredictable

Unpredictable

Chaos

- Unclear
- Unpredictable
- Unstable

Unstable

Chaos

- Unclear
- Unpredictable
- Unstable
- Adventurous

and highly adventurous.

You'll see the reason for that in a minute

let me first let you show **how many different solutions** we've already seen

and **how wide their variety** is.

We saw ...

We have seen things like ...

We saw ...

- Ruby Gettext

Ruby Gettext – the dinosaur and grandfather of Internationalization

We saw ...

- Ruby Gettext
- DHH's Localization

David's original Localization plugin as well as ...

We saw ...

- Ruby Gettext
- DHH's Localization & Thomas' Localization

Thomas Fuchs' Localization plugin – both very **simple**

We saw ...

- Ruby Gettext
- DHH's Localization & Thomas' Localization
- GLoc

GLoc – **one of the first** bigger implementations

We saw ...

- Ruby Gettext
- DHH's Localization & Thomas' Localization
- GLoc
- Globalize

Globalize - the only solution with solid Model translations

We saw ...

- Ruby Gettext
- DHH's Localization & Thomas' Localization
- GLoc
- Globalize & Globalite

Globalite – a very slick solution by **Matt Aimonetti**

We saw ...

- Ruby Gettext
- DHH's Localization & Thomas' Localization
- GLoc
- Globalize & Globalite
- Simple Localization

Simple Localization – also a very interesting solution by **Stephan Soller**

Both have contributed a lot to the Rails I18n project

We saw ...

- Ruby Gettext
- DHH's Localization & Thomas' Localization
- GLoc
- Globalize & Globalite
- Simple Localization
- Gibberish

Gibberish – maybe the most recent full solution

Came along as the **cool kid on the block**

... and many more.

... as well as ...

There are solutions for special needs like ...

... as well as ...

- Localizing Rails to certain Locales

Localizing Rails to **certain languages**

like Swedish, Brazilian, Korean and so on

... as well as ...

- Localizing Rails to certain Locales
- Plugins adding to other plugins

There are plugins that **add functionality** to other plugins

For example there is something that adds a **Javascript Layer** on top of Gibberish

... as well as ...

- Localizing Rails to certain Locales
- Plugins adding to other plugins
- Tools on top of all of this

And things that just adds separate, useful tools

... and more.

stuff like that ... of course there’s much more.

I’ve just wanted to give you an impression of these things.

Problems

So when you take into account **how complex our languages** are ...

and **how many funny problems** there are to solve ...

and we have **lots of lonely programmers** working on this ...

and everybody **focusses** on the features he actually needs himself ...

what do you think will you get?

Problems

- Isolated, incompatible solutions

What you get is a huge mess.

We have lots of isolated, incompatible solutions.

Problems

- Isolated, incompatible solutions
- Lonely developers & bad support

Things like support and documentation

just don’t get better

with a lonely developer solving their own needs and

then – of course – often walking away and focussing on

completely different things.

Problems

- Isolated, incompatible solutions
- Lonely developers & bad support
- Galores of Monkeypatches

Then

all of our solutions basically were

huge Monkeypatches to Rails

with the usual problems resulting from that.

Problems

- Isolated, incompatible solutions
- Lonely developers & bad support
- Galores of Monkeypatches
- Wheels continuously re-implemented

And all of us re-implemented the wheel in one way or the other.

So, in a way, we wasted lots of resources.

But - on the other hand - this situation was actually necessary for us

to experiment with solutions and learn how Rails Internationalization could be done.

Rails I18n project

Because of this situation ...

Rails I18n project

- Started in September 07

... last year in September a **group of developers** from **several Rails Internationalization plugins** gathered for a chat session ...

Rails I18n project

- Started in September 07
- Common goal

... we agreed that we wanted to implemented a **Rails core patch**

which should **solve the problems** we had with this stuff

Rails I18n project

- Started in September 07
- Common goal
- Invited everybody

We **invited developers** from all existing solutions and

started **meeting regulary**,

collected requirements,

discussed our ideas ...

Rails I18n project

- Started in September 07
- Common goal
- Invited everybody
- Much upfront discussion

Actually there was really **a lot** of discussion and

initially it was **hard to agree on anything**.

You know that sort of situation.

Everybody wants to get their **own favorite little features** to the table.

Everybody has **different beliefs of what's good** and what's necessary.

So this discussion went on **endlessly**.

And in the beginning of 2008 our **work slowed down**

apparently we really needed a **creative break** and some time to rethink everything.

Rails I18n project

- Started in September 07
- Common goal
- Invited everybody
- Much upfront discussion
- Implementation in May 08

Then, in May 2008 we got **back to the drawing board** with a fresh approach

...

Implementation

and now we were able to agree on

a very **slim and lightweight implementation**

that covered all of our **requirements**

Implementation

- Ruby gem

It works as a **Ruby gem**

That means it is not only **suited for use in Rails**

but also other frameworks

Implementation

- Ruby gem
- Minimum changes in Rails

We wanted only **minimal changes** in Rails

Implementation

- Ruby gem
- Minimum changes in Rails
- Minimum additional load in Rails

We wanted it to add **as less load** to Rails as possible

Implementation

- Ruby gem
- Minimum changes in Rails
- Minimum additional load in Rails
- Common API for everybody

And of course we still wanted it to implement

a **common API** where everybody could build on.

Solved!

So, at this point Jeremy Kemper didn't know about our work

but in a RailsConf Portland RailsEnvy interview

he predicted for Rails 2.2 that ...

Solved!

... Internationalization will be solved.

Solved!

Jeremy said that was a joke ...

When we asked Jeremy about this

he said it was a joke

Solved!

Please crack more of them!

So we think

Jeremy should definitely crack more jokes.

In the end this joke actually got us going to

finally propose our work to the core team.

Key decisions

In hindsight there were a few key decisions

Before we managed to agree on these

we had not been able to get **anything** really done.

These key decisions were ...

Key decisions

- 100% swappable

We agreed to make the actual implementation of all the logic exchangeable.

So you can always plug in something more powerful if you need to.

Key decisions

- 100% swappable
- Do the simplest thing that ever could work

We radically sticked to the principle of "doing the simplest thing that ever could work" for Rails

Key decisions

- 100% swappable
- Do the simplest thing that ever could work
- Do not aim for a full-blown L10n solution

And maybe most importantly ...

And we **just stopped aiming** for a solution

that did **anything** else than the bare minimum

that's needed for Rails.

Rails I18n

So, what we now ship with Rails is ...

Rails I18n

Powerful I18n API

a **common API** for Internationalization

which is basically just a bunch of definitions how things are supposed to work

So this is just a Ruby module with some public methods on that

Rails I18n

Powerful I18n API

+

Simple Backend (en-US only)

These methods **delegate** to a backend that **actually implements** the API.

The backend that is shipped with Rails is intentionally named the **Simple** backend

And the **only requirement** for the Implementation in the Simple backend is

that it works for American English

because that's what we need for Rails to keep it working like it worked before

Rails I18n

```
Rails.localize 'en-US'
```

That means: the **most important parts** that previously were **hardcoded** right into Rails

are **now abstracted** out of the code and

moved to a simplistic **backend**

accessible through just a **few public methods**.

So this ...

Rails I18n

```
Rails.localize 'en-US' #=> true
```

... is true.

Rails I18n

```
Rails.localize 'en-US' #=> true
Rails.localize :other
```

And this ...

Rails I18n

```
Rails.localize 'en-US' #=> true
Rails.localize :other  #=> maybe!
```

... **might** work.

Maybe you **can** localize Rails to your **favorite language** if it is very **similar to English**.

There are quite some resources out there that show how to do this.

For everything else we still need to **add plugins** or other libraries.

Rails Translations

- Shipped as YAML files
- You can also use plain Ruby files

For Rails itself we settled with shipping the translations as YAML files

And with stock Rails you can also use Ruby files

so you can have lambdas in your translation data.

I18n API

What's in the box?

Ok,

so I'll run through the main features that this API provides.

API methods

`I18n`

As I said the API is provided as a Ruby module

with two main methods on it.

These are:

API methods

```
I18n.translate
```

I18n.translate

which you can use to look up translations from the backend

and for convenience it has an alias

API methods

```
I18n.t :rails
```

just #t

you need to pass it at least one parameter

which is the key

So this would look up the translation that's stored for the key :rails in the current locale, for example, American English

API methods

```
I18n.t :rails

I18n.localize
```

The other method that you'll use regularly is localize.

This one basically allows you to format Date and Time objects for a certain locale.

Again, for convenience there's an alias ...

API methods

```
I18n.t :rails

I18n.l Time.now
```

of course that's #l

and you pass it the object that you want to localize

Lookups

So the #translate gives you very powerful means to look up translations.

Lookups

- Flexible keys

```
I18n.t :rails
```

So you can look up your translations using a Symbol as a key

like this

and this is what we'd generally encourage you to do

Lookups

- Flexible keys

```
I18n.t :rails
I18n.t "rails"
```

but you can also use a String

Lookups

- Flexible keys
- Scopes (aka namespaces)

Also, you have scopes.

So you can namespace your translations and structure them in a reasonable way.

Lookups

- Flexible keys
- Scopes (aka namespaces)

```
activerecord:
```

For example in the ActiveRecord translations YAML file ...

there's a scope activerecord ...

Lookups

- Flexible keys
- Scopes (aka namespaces)

```
activerecord:
  errors:
```

and then there's a scope errors

Lookups

- Flexible keys
- Scopes (aka namespaces)

```
activerecord:
  errors:
    messages:
```

a scope messages

Lookups

- Flexible keys
- Scopes (aka namespaces)

```
activerecord:
  errors:
    messages:
      invalid: "is invalid"
```

and then there's finally the translation for the key :invalid

so scopes are basically groups or namespaces of translations

or you can think of them as nested Hashes

Lookups

- Flexible keys
- Scopes (aka namespaces)
- Namespace lookup

```
I18n.t :"activerecord.errors.messages"
```

Consequently there's a way to look up whole namespaces – or groups – of translations

For example to look up all error messages in ActiveRecord

this ...

will give you a Hash with all those messages

Lookups

- Flexible keys
- Scopes (aka namespaces)
- Namespace lookup
- Bulk lookup

```
I18n.t [:ruby, :rails]
# => ["Ruby", "Rails"]
```

Then there's bulk lookup.

That just means that you can pass an Array of keys you want translated

and it will return you a corresponding array of translations

Lookups

- Flexible keys
- Scopes (aka namespaces)
- Namespace lookup
- Bulk lookup
- Defaults

And finally, of course, there's defaults

so you can specify what happens when a translation can not be found

Lookups

- Flexible keys
- Scopes (aka namespaces)
- Namespace lookup
- Bulk lookup
- Defaults

```
I18n.t :rails, :default => "rails"
```

for example when you give it a String as a default and

the translation can not be found it just returns the String

Lookups

- Flexible keys
- Scopes (aka namespaces)
- Namespace lookup
- Bulk lookup
- Defaults

```
I18n.t :rails, :default => :python
```

But when you give it a Symbol

And then it will translate this Symbol and give you that translation as a default

Lookups

- Flexible keys
- Scopes (aka namespaces)
- Namespace lookup
- Bulk lookup
- Defaults

```
I18n.t :rails, :default => [:python, "php"]
```

And finally you can specify an array of Symbols and/or Strings and

it will just check everything in that order and return the first value that's not nil

Interpolation

Interpolation is another feature.

That means you want to abstract your translations in certain situations ...

Interpolation

```
message: "Thanks {{name}}!"
```

For example you have a translation for the key :message

And there's variable :name in that ...

Interpolation

```
message: "Thanks {{name}}!"

I18n.t :message, :name => "Jeremy"
```

Then you can pass the value of that variable to #translate ...

Interpolation

```
message: "Thanks {{name}}!"

I18n.t :message, :name => "Jeremy"

# => "Thanks Jeremy!"
```

and it will parse the value into the translation

Like this

This looks trivial but it's actually a very important feature

because concatenating translations is kind of an anti-pattern in most situations.

Pluralization

```
# en-US
:days => {
  :one   => "one day"
  :other => "{{count}} days"
}
```

Also, the API provides means to define Singular and Plural translations of course

For example you could have a Singular and a Plural translations like this

Pluralization

```
# en-US
:days => {
  :one   => "one day"
  :other => "{{count}} days"
}

I18n.t :days, :count => 1
```

Then you can pass a count option to #translate

Pluralization

```
# en-US
:days => {
  :one   => "one day"
  :other => "{{count}} days"
}

I18n.t :days, :count => 1 # => "one day"
```

So in this case it would pick the Singular translation because count is 1

Pluralization

```
# en-US
:days => {
  :one   => "one day"
  :other => "{{count}} days"
}

I18n.t :days, :count => 1 # => "one day"
I18n.t :days, :count => 2
```

And when you pass it 2

Pluralization

```
# en-US
:days => {
  :one   => "one day"
  :other => "{{count}} days"
}

I18n.t :days, :count => 1 # => "one day"
I18n.t :days, :count => 2 # => "2 days"
```

it would pick the Plural translation

I18n API

You can look up everything online :-)

So that’s it already – all very basic stuff of course.

Backends

Plug in what you need

Like I said, the I18n API ships with a Simple backend ...

The Simple backend

only translates to en-US

... which implements all of these features for American English

So for example you can’t use it to do ...

Czech pluralization

... pluralization for Czech

because they have different rules for pluralization

Czech pluralization

```
# en-US
:messages => {
  :one   => "one message"          # count == 1
  :other => "{{count}} messages" # count != 1
}
```

As I just showed you

your English translations include something like this ...

Czech pluralization

```
# cs-CZ
:messages => {
  :one   => "jedna zpráva"
  :few   => "{{count}} zprávy"
  :other => "{{count}} zpráv"
}
```

But in Czech you have two plural forms – one of them is called **Paucal**

Czech pluralization

```
# cs-CZ
:messages => {
  :one   => "jedna zpráva"     # 1
  :few   => "{{count}} zprávy"
  :other => "{{count}} zpráv"
}
```

they use the first one as a singular just like one does in English

Czech pluralization

```
# cs-CZ
:messages => {
  :one   => "jedna zpráva"      # 1
  :few   => "{{count}} zprávy" # 2..4
  :other => "{{count}} zpráv"
}
```

and they use the Paucal, the second form,

for everything between 2 and 4

Czech pluralization

```
# cs-CZ
:messages => {
  :one   => "jedna zpráva"      # 1
  :few   => "{{count}} zprávy" # 2..4
  :other => "{{count}} zpráv"  # >= 5
}
```

and the actual plural for everything else.

Swap the backend

So the Simple backend can not do that.

But we can easily swap that

and the Internationalization API actually encourages you to do so.

Swap the backend

```
module I18n::Backend
  class CzechPlurals < Simple

  end
end
```

So how could you do that?

You would just **create a class** for your custom backend and

maybe extend the Simple Backend

Swap the backend

```
module I18n::Backend
  class CzechPlurals < Simple
    def self.pluralize(*args)
      # your logic
    end
  end
end
```

implement your custom logic

Swap the backend

```
module I18n::Backend
  class CzechPlurals < Simple
    def self.pluralize(*args)
      # your logic
    end
  end
end
I18n.backend = I18n::Backend::CzechPlurals.new
```

and tell the I18n module to use it.

That's it.

Basically this is what **future plugins will do** a lot.

For example there definitely will be a backend that stores translations to the **database**,

and I'm sure there's one soon to store them to **Gettext** files.

Demo

Short

So let’s have a really brief look how you can use this in your application

Today's landscape

Scenarios: What can we use today?

So, if you're actually working on an application today, what can use use?

Of course this really depends on your requirements.

Let's have a look at some scenarios ...

Static translations

- Plain Rails I18n + Simple backend
- Look at demo apps on GitHub

Let's say you just need a **few static translations** and maybe some date and number formats.

Then you can easily **reuse the Simple backend** that Rails ships with.

By now there are a couple of **demo applications** on GitHub that show how to do that.

Fancy pluralizations

- Experimental Pluralizing backend
- Custom pluralizers

Let's say you have a language that requires different pluralization rules.

Then there's an experimental Pluralizing backend for that where you can hook in pluralization logic as lambdas.

Fancy localizations

- Experimental Chain backend
- Custom localization backends
- New: http://github.com/yaroslav/russian

Let's say you need to translate your application to Russian, or Czech or another language that has some more complex date formats.

For that there's an experimental Chain backend where you could hook in a custom backend tailored for just these languages' date formats.

Also there's already a full plugin solution by Yaroslav Markin from Russia that implements things like these and I think it's easy to adapt it for other languages.

Model translations

- Globalize 2
- Now 200% more slick & unobtrusive
- Alternative: see Gist on http://rails-i18n.org

If you need model translations there'll be Globalize2.

Globalize2 will look completely different to Globalize. It will be pretty lightweight and unobtrusive. Joshua will tell you more about that in the interview.

Also there's an alternative approach, that's linked on rails-i18n.org

The future of Rails I18n

So, if we **still** have to rely on external code, like plugins, to localize our applications.

What do we win?

What do we win?

The short answer to that is ...

A lot!

A lot!

What do we win?

- No monkeypatching

First of all, we don't need **no monkeypatching** any more.

Well, ...

What do we win?

- ~~No~~ monkeypatching

actually of course we still **will monkeypatch** Rails to experiment with new features.

What do we win?

- Less monkeypatching

But we definitely won't need to do probably 99% of what we've done before **all the time**

like translate **error messages**,
translate **all the helpers** that Rails provides,
or anything like that

What do we win?

- Less monkeypatching
- Common API

Also, because we now have a **common API**

things will be much **better exchangeable**.

What do we win?

- Less monkeypatching
- Common API
- Common locale data

We will aggregate **common locale data** over time

So we will build up a **Rails specific** pool of Localization **knowledge**.

What do we win?

- Less monkeypatching
- Common API
- Common Locale data
- Atomic plugins

Also we hope that plugins will be much more **focussed** on implementing certain **atomic features**.

This is actually something **we encourage everybody** to do.

So far, you could not possibly **combine** three, four, five **different plugins** and expect anything to work.

We hope we're now able to do a better job here in future.

Dev.cycle.next.start!

So, of course,

with all of this stuff now released

our next development cycle has started

... right now.

Next cycle

- Implement atomic plugins, backends, ...

Of course we're all going to implement our stuff next.

In fact by now there are **already quite some useful things** out there.

Next cycle

- Implement atomic plugins, backends, ...
- Collect

During this time we will try to stay in touch with each other ...

and then, again, ...

collect everything ...

Next cycle

- Implement atomic plugins, backends, ...
- Collect
- Review

... review it ...

Next cycle

- Implement atomic plugins, backends, ...
- Collect
- Review
- Discuss

... again, discuss it ...

Next cycle

- Implement atomic plugins, backends, ...
- Collect
- Review
- Discuss
- Patch Rails core

... and finally aim for another core patch.

The reason why we want to first collect everything and look at it as a whole

and **then** move it to Rails as a rather **complete package**

is that many of these things **make most sense in context** with each other.

Next cycle

- Implement atomic plugins, backends, ...
- Collect
- Review
- Discuss
- Patch Rails core

We hope this will then **result** in ...

Future landscape

... a landscape of Rails Internationalization solutions where we have ...

Future landscape

- Toolbox of tiny plugins

... a distributed toolbox of focussed plugins ...

Future landscape

- Toolbox of tiny plugins
- Less bloated Jacks-of-all-trades

... rather than those huge solutions trying to solve everything under the sun at once ...

Future landscape

- Toolbox of tiny plugins
- Less bloated Jack-of-all-trades
- More compatible solutions

We'll have more compatible solutions

Future landscape

- Toolbox of tiny plugins
- Less bloated Jack-of-all-trades
- More compatible solutions
- Common data

... maybe the most exciting thing is

that we will get to have **common locale data**.

So ...

Globalize 2

<insert interview with Joshua here>

http://rashgash.com

Let's see what Joshua Harvey has to say about Globalize 2.

By the way the guitar shop that you can see in the video is actually the shop rashgash.

The website at rashgash.com is the reason why Joshua initially wrote Globalize as he mentions in the interview.

Globalize 2

- Model translations: unobtrusive
- View translations
- RFC4646/47 compliance
- Locale fallback
- load_path for loading translations
- everything decoupled and reusable

So this is a brief overview of features that Globalize 2 will provide.

Of course there's model translations - kind of Globalize's USP in the past.

For View translations we'll basically use the Rails I18n API

There will be some useful tools like standards compliance tool for locales

which we'll need for locale fallbacks and stuff like that.

A locale load_path tool that helps loading translation data more easily.

Globalize 2

Thank you, sponsors!

Actually the implementation of Globalize 2 is sponsored.

Globalize 2

Thanks, sponsors!

*BEST*group Consulting & Software, Berlin
http://www.best-group.eu

Metaversum
http://metaversum.com

By the company BESTGroup Consulting & Software from Berlin.

So, THANK YOU, BESTGroup for making this possible.

Also, Metaversum, a company also from Berlin will start sponsoring us right after this conference.

So, thank you, Metaversum in advance.

Rails I18n + Globalize 2

We're looking for sponsors!

*BEST*group Consulting & Software, Berlin
http://www.best-group.eu

Metaversum
http://metaversum.com

And actually we are looking for more sponsors

not only for the Globalize project but also for the Rails I18n project

because we really want to

put this project on a really solid foundation

for example we want to have somebody working on stuff like documentation on a regular basis

and we want to provide better support for newbies

Rails I18n + Globalize 2

We're looking for sponsors!

*BEST*group Consulting & Software, Berlin
http://www.best-group.eu

Metaversum
http://metaversum.com

So if you're interested in sponsoring a

really cutting edge and high-profile

Ruby on Rails Open Source project

that already gets tons of attention

and will get even more attention in the next couple of months

just catch me afterwards and we can talk about that.

Contribute!

What can you do to help?

So, how can everybody in this room get involved?

Contribute!

- Use this stuff and report back

Of course everybody can just start playing with all of this,

register to our mailinglist and give feedback.

We need tons of that.

Contribute!

- Use this stuff and report back
- Share your translations

You can share your translations.

We already have a repository on GitHub for that, so if you've done translations for Rails please submit them.

Contribute!

- Use this stuff and report back
- Share your translations
- Build focussed plugins and share them

You can build plugins focussed on certain features that are still missing.

And please don’t forget to pitch them on the mailinglist and our wiki.

Contribute!

- Use this stuff and report back
- Share your translations
- Build focussed plugins and share them
- Help improving Rails integration

You can help improving the Rails integration.

You know Rails helpers are there and they are working.

But that doesn’t mean they’re already perfect regarding Internationalization.

So you can help to improve that.

Contribute!

- Use this stuff and report back
- Share your translations
- Build focussed plugins and share them
- Help improving Rails integration
- Sponsor our work

And of course, you can sponsor our work

so we can invest some money in things that are hard to do otherwise.

Resources

see http://rails-i18n.org

So, basically this was my talk ...

We've collected pretty much everything I mentioned in this talk as well as lots of other resources

on rails-i18n.org

So, look it up

And ...

Thanks!

Matt Aimonetti	Clemens Kofler	Julian Tarkhanov
Sven Fuchs	Marko Seppä	Simon Tokumine
Joshua Harvey	Karel Minarik	Vladimir Stiff
Saimon Moore	Trevor Turks	Glenn Pow
Stephan Soller	Yaroslav Markin	Tore Darell
Jeremy Kemper	Luca Guidi	Sam Lown
Iain Hecker	Joshua Sierles	Akira Matsuda
Fabio Akita	Diego Carrion	Ingo Weiss

... and many more ...

Thanks to everybody who contributed!

Questions?

Ok, so ...

what are all those questions you want to ask?