The Definitive Guide to Jakarta Faces in Jakarta EE 10

Building Java-Based Enterprise Web Applications

Second Edition

Bauke Scholtz
Arjan Tijms

Apress®

The Definitive Guide to Jakarta Faces in Jakarta EE 10

Building Java-Based Enterprise Web Applications

Second Edition

Bauke Scholtz
Arjan Tijms

Apress®

The Definitive Guide to Jakarta Faces in Jakarta EE 10: Building Java-Based Enterprise Web Applications

Bauke Scholtz
Willemstad, Curaçao

Arjan Tijms
Amsterdam, Noord-Holland, The Netherlands

ISBN-13 (pbk): 978-1-4842-7309-8
ISBN-13 (electronic): 978-1-4842-7310-4
https://doi.org/10.1007/978-1-4842-7310-4

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Laura Berendson
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Shutterstock (www.shutterstock.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Authors

Bauke Scholtz is an Oracle Java Champion, a committer of the Jakarta Faces API and Mojarra projects, and the main creator of the Jakarta Faces helper library OmniFaces. He is on the Internet more commonly known as BalusC who is among the top users and contributors on Stack Overflow. Bauke has integrated several OmniFaces solutions into Jakarta Faces. He is a web application specialist and consults or has consulted for Virtua Inc, Mercury1 Limited, MyTutor, Nava Finance, LinkPizza, ZEEF, M4N/Zanox, ITCA, RDC, and clients from fintech, affiliate marketing, big data, social media, and more as part of his more than 20 years of experience. This book offers Bauke the opportunity to go into depth as to answering most frequently asked questions and correctly solving most commonly encountered problems while using Jakarta Faces.

Arjan Tijms works for Payara Services Ltd and is a Jakarta Faces (JSR 372) and Security API (JSR 375) Expert Group member. He is the co-creator of the popular OmniFaces library for Jakarta Faces that was a 2015 Duke's Choice Award winner and is the main creator of a set of tests for the Jakarta EE authentication SPI (JASPIC) that has been used by various Jakarta EE vendors. Arjan holds an MSc in Computer Science from the University of Leiden, the Netherlands. Writing about this topic was a natural choice for Arjan; he has already written much about it at his blog and would like to expand that by contributing to a book.

About the Technical Reviewer

Luqman Saeed is a Jakarta EE developer with Pedantic Devs. He has been in software development for close to a decade. He started with PHP and now does Jakarta EE full time. His goal on Udemy is to help you get productive with powerful, modern, intuitive, and easy-to-use Jakarta EE APIs. He will provide you with the best of vanilla, pure, and awesome Jakarta EE courses to help you master the skills needed to solve whatever development challenge you have at hand.

CHAPTER 1

History

This chapter describes the history of Jakarta Faces, starting from its early conception and ending where we are today at the moment of writing. We'll discuss how the Jakarta Faces API (application programming interface) itself evolved, which important events took place during that evolution, and who some of the people were that were involved in all of this.

This is in no way a complete description of the history, and the reader should take notice of the fact that many more events took place and many more people were involved than we were able to mention here.

In the Beginning…

Jakarta Faces goes back a long time. Its initial JSR, JSR 127, started in 2001. At that time, the Struts web framework was wildly popular, although it wasn't that long ago that it was released itself (around 2000). Despite Struts' popularity, a large number of other web frameworks were in use in the Java space, and new ones were popping up all the time. Jakarta Faces was conceived as an attempt to bring a standardized MVC (model-view-controller) web framework base into the overall Jakarta EE platform.

Controversies are quite common in the web framework space, and Jakarta Faces is no exception here. Right at the start of its inception, there was a big controversy where Apache opposed the creation of Jakarta Faces on the bases that Apache Struts already existed and a closed source alternative would have little value. Apache therefore voted against the creation of Jakarta Faces with the following comment:

> *This JSR conflicts with the Apache open source project Struts. Considering Sun's current position that JSRs may not be independently implemented under an open source license, we see little value in recreating a technology in a closed environment that is already available in an open environment.*

© Bauke Scholtz and Arjan Tijms 2022
B. Scholtz and A. Tijms, *The Definitive Guide to Jakarta Faces in Jakarta EE 10*,
https://doi.org/10.1007/978-1-4842-7310-4_1

> *To the extent that this JSR extends beyond Struts today, we would encourage the Sun developers proposing this JSR to join the Sun developers already leading Struts to create an open solution at Apache, something which when finished would be assured of being able to be implemented as open source.*

Eventually the conflict was resolved when after about a year into the process, spec lead Amy Fowler (from Swing fame) was replaced by Craig McClanahan, the very father of the Struts project that Jakarta Faces was said to be competing with. The open source restriction was lifted as well, and the open source Jakarta Faces implementation, called MyFaces, was developed in parallel with the (then nameless) RI and hence the specification itself. MyFaces initially started as an LGPL licensed project at `http://sourceforge.net` in December 2002 and had an initial 0.1 release conforming to what was then called an "Early Access Specification" in January 2003.

Open source implementations are the most common implementations in Jakarta EE 8, and there's barely any EE specification at the time of this writing (2018) that's still implemented as closed source. In 2001, however, this was not just uncommon; it was actually not allowed for new JSRs. Allowing for an open source implementation was therefore quite a change, and the honor fell to Jakarta Faces to be the first of its kind for which this was allowed.

Despite the open source implementation being allowed, the actual development of the spec was still done in secret and behind closed doors. There was no public mailing list and no tracker (e.g., a JIRA instance) for the public to create issues or express wishes. Occasionally, interviews were being done, and in the fall of 2002, by then, former spec lead Amy Fowler did reveal quite a few details about Jakarta Faces, but largely the project was shrouded in mystery for the general public.

The team behind Jakarta Faces was, however, hard at work. The first email to the internal JSR 127 list was sent on August 17, 2001. As with most projects, the team spent the initial months on gathering requirements and looking at the existing competing products. A package name was chosen as well. The initial placeholder package, which was "`javax.servlet.ui`", now "`javax.faces`", was chosen as the package to use. The very first technical architecture to be considered was the component model. For a component-based MVC framework, this is obviously one of the most important aspects. During the last month of 2001 and the first two months of 2002, the team looked at what is now known as the Managed Bean (called "Object Manager" then). Managed beans with their scopes, names, and dependency injection are clearly another cornerstone of the Jakarta Faces framework. Events and the model behind it were being looked at as well during that time frame.

In the second quarter of 2002, two other cornerstones of Jakarta Faces were discussed: the Expression Language (inspired by Jakarta Tags), which is instrumental for the so-called bindings of beans from a template to backing code, and the factory finder, which allowed key parts of Jakarta Faces to be replaced and, although perhaps not fully realized at the time, may have contributed greatly to Jakarta Faces still being relevant some 16 years later.

It was in this same quarter that Craig McClanahan took over as spec lead, father of Struts and architect of Tomcat's Servlet container. Not long after the discussion about using Jakarta Pages started, a discussion, perhaps unbeknownst to the team at the time, that would, unfortunately, have a rather negative impact on Jakarta Faces later on. Around the end of the year 2002, Ed Burns, who like McClanahan had also worked on Tomcat before, joined the team as co-spec lead. Burns is the person who would eventually become the main spec lead of Jakarta Faces for well over a decade.

While the team continued to work on things like the aforementioned managed beans and the so-called value binding, which is the Java representation of the also aforementioned expression language binding, the first dark cloud appeared when in the spring of 2003, team member Hans Bergsten realized that there were very real and major issues with using Jakarta Pages as the templating language for Jakarta Faces. He brought these concerns to the team, but ultimately they weren't addressed and instead the following months were spent, among other things, on a variant of the value binding; it later on became clear that the method binding and the state saving mechanism were another of Jakarta Faces's less-than-ideal implementations.

Jakarta Faces 1.0 and its still nameless RI were eventually released on March 11, 2004—coincidentally, a mere two weeks before the release of another framework that's still strong today, Spring 1.0. MyFaces released its 1.0.0 alpha version only days later, on March 19. It's perhaps an interesting observation that Jakarta Faces went final with a full-fledged XML-based dependency injection (DI) framework just before Spring, which is largely known for its DI, went final.

Jakarta Faces 1.0 was generally well received; despite a rather crowded market with competitors such as Tapestry, WebObjects, Velocity, and Cocoon operating, not less than three books from writers such as Horst Caymann and Hans Bergsten appeared in the months after, and the eXo platform (a Digital Collaboration Platform) started using Jakarta Faces right away.

Hans Bergsten's earlier concerns, however, became painfully clear almost just as quickly; the Jakarta Pages technology is based on processing a template from start to end, immediately writing to the response as tags are encountered. Jakarta Faces, however, requires a phased approach where components need to be able to inspect and act on the component tree, which is built from the tags on the page, before starting to write anything to the response. This mismatch led to many strange issues, such as content disappearing or being rendered out of order.

Only three months after the introduction of Jakarta Faces, Hans Bergsten made a strong case of dropping Jakarta Pages in his legendary article “Improving Jakarta Faces by Dumping Jakarta Pages.” There, Bergsten explains how ill-suited Jakarta Pages is for use as a template language in Jakarta Faces, but he also presents a glimmer of hope; because of Jakarta Faces’s great support for extendibility, it’s relatively easy to introduce alternative templating simply by replacing the so-called view handler, something which Jakarta Faces explicitly allows. It would, however, take five long years until Jakarta Faces would indeed ship with a more suitable view templating language, and even though Jakarta Pages had been essentially deprecated at that point, it’s still present in Jakarta Faces at the time of writing.

The Adolescent Years

Back in 2004, another first befell Jakarta Faces; on June 28, Ed Burns announced that the source of the RI was released by Sun. This represented a major milestone as before that date, most technology in active use by Sun was closed source. Initially, the source was licensed under the somewhat exotic JRL, but later this would be changed to dual licenses, GPL with classpath exception and CDDL. At the same time as this announcement, the tradition was established that every new feature or bug fix should be accompanied by a test and that all existing tests should be executed before committing the change. Some 14 years later, there’s a largely different set of people working on the RI source, and the project structure and code conventions have changed as well, but the test-driven tradition is still being uphold in its original form.

At that point, Ed Burns decided to focus more on the specification aspects of Jakarta Faces as the Jakarta Faces 1.2 spec work had started right away, and Jayashri Visvanathan, one of the early team members, took on the lead role concerning the implementation aspects, with Ryan Lubke, working as the TCK (testing) engineer.

Still only a few months old, a variety of component libraries for Jakarta Faces had already started to pop up, although all of them were commercial. Among those was the one from Oracle, ADF Faces. ADF Faces was put on Oracle’s road map well before Jakarta Faces 1.0 went final, and the first early access release was presented on August 17, 2004. Its lead was Adam Winer, who represented Oracle in the team that created

Jakarta Faces 1.0. ADF Faces primarily contained a set of rich components, but also a dialog framework, and remarkably already featured partial page rendering (PPR), quite a bit ahead of the later crop of AJAX solutions. ADF Faces also contained a "for each" tag (`af:forEach`) that actually worked. Adam Winer explained in these early days that such tag is not quite trivial to build but promised that Oracle would contribute the knowledge back to Jakarta Faces itself.

The ADF Faces components originated mostly from the earlier User Interface XML (UIX) framework, of which Adam Winer was the lead architect as well. Earlier versions of UIX used the names "Cabo," "Baja," and "Marlin." UIX was a rich client framework for use in the browser. With Jakarta Faces sharing more than a few similarities to UIX and with its lead, Adam Winer, being part of the original Jakarta Faces team, it's perhaps not unreasonable to surmise that UIX influenced Jakarta Faces. Such similarities include the concept of components with separate renderers, Jakarta Pages tag handlers, and declarative options to compose a page and the ability to instantiate those same components programmatically in Java. There was even a conceptually similar data binding, although with a less elegant syntax. Instead of, say, `value="#{user.age}"`, UIX would use `data:value="age@user"` but also required a kind of producer to be defined on each page to declare where "`user`" comes from and then nest the page's content within that declaration. By contrast, Jakarta Faces and EL have always used global definitions and left it up to the user to avoid name clashes.

One of the first, if not the first open source component library in 2004, was Matthias Unverzagt's OurFaces. As Jakarta Faces did not have its own resource API (application programming interface) at the time to serve up things like images, OurFaces required a Servlet to be added to `web.xml`, the so-called `SkinServlet` (`ourfaces.common.webapp.SkinServlet`). The significance of this is that it became a rather common thing for Jakarta Faces libraries in those days to ask their users: add something manually to `web.xml` before the component library can be used.

Most of the last months of 2004 and early months of 2005 were spent by the Jakarta Faces 1.2 expert group (EG) working on various Jakarta Pages and EL issues, such as the Jakarta Tags `<c:forEach>` support and the generation of IDs in Jakarta Pages, as well as on the dreaded "content interweaving" issue, which refers to the aforementioned content that appears at wrong places in the response when rendering.

While OurFaces may have been one of the first component libraries, it didn't last, and few will remember it or have even heard about it today. This is not quite the same for another framework that has its roots in early 2005, namely, Alexander Smirnov's Telamon framework, later renamed Ajax4jsf. This framework was one of the first of its kind that combined Jakarta Faces and the then new and fresh AJAX technology. The beauty of Ajax4Jsf was that it could add AJAX support to existing components, which weren't built with AJAX support in mind at all by enclosing them among others in the `<a4j:region>` tag. This technology was incorporated in the Exadel Visual Component Platform, which was released in March 2006 and would later be renamed RichFaces and would become one of the most memorable Jakarta Faces component libraries.

At around the same time Alexander Smirnov started work on what eventually would become RichFaces, a company called ICEsoft started working on a Jakarta Faces component library. ICEsoft had been in business for a couple of years and had been working on a product called ICEbrowser, a Java-based browser, and a product called ICEbrowser beans, which were "lightweight, configurable JavaBean components that can be rapidly integrated into Java client applications." During JavaOne 2005 of that year, on June 27, ICEsoft announced its component library for Jakarta Faces—ICEfaces. This was based on AJAX as well but incorporated AJAX directly into the components. ICEsoft called its specific technique "patent pending Direct-to-DOM," which basically meant that changes coming from the server were directly injected into the DOM tree structure of a web page. A final version wasn't available right away though, but an early access release was provided. This was closed source but cost-free.

Meanwhile, Jakarta Faces EG member Jacob Hookom, inspired by Hans Bergsten's concerns about the unsuitability of Jakarta Pages, grabbed the bull by the horns and started working himself on that alternative templating language envisioned by Bergsten. In August 2005, this work had progressed into a usable initial version. The name of this templating language? Facelets! It immediately took the Jakarta Faces world by storm. Kito Mann published the first part of a series of articles about it on JSFCentral the very first month, and Richard Hightower published the famous article "Facelets fits Jakarta Faces like a glove" several months later.

Oracle had not been sitting still either in 2005, and after about 16(!) early access releases, it announced in late 2005 at the JavaPolis conference in Antwerpen (nowadays called Devoxx) that ADF Faces would be donated to MyFaces and thus become open source.

In the first month of 2006, Jacob Hookom and Adam Winer contemplated the terrible implementation of Jakarta Faces's state save mechanism. This worked by first creating a component tree from a template and then, near the end of the request, blindly serializing the entire tree with all data that may have been put there during the request. During a postback, the tree is restored from this serialized form (hence the name of the phase "restore view"). This is a tremendous waste, as the majority of this information is already available in the template. Especially when doing AJAX requests with client-side state saving, this poses a very big burden, but it is also a problem when storing this state on the server as it massively increases Jakarta Faces's memory usage. One of the main reasons for doing state saving in such terrible way again has to do with that one decision: to support Jakarta Pages. With Jakarta Faces 1.2 about to go final, there was unfortunately no time left to fix this for version 1.2.

Even though it was clear at this point that Facelets was the future of Jakarta Faces, when Jakarta Faces 1.2 was eventually released in May 2006, it still contained only Jakarta Pages. Not all was bad though. Thanks to a cooperation between the Jakarta Faces and Jakarta Pages EGs, a revision of Jakarta Pages was released, Jakarta Pages 2.1, which was much better aligned with the demands of Jakarta Faces. On top of that, Jakarta Pages's expression language and Jakarta Faces's expression language were merged. The result was UEL (Unified Expression Language). A very practical advantage of UEL is that Jakarta Faces components no longer have to convert Strings manually into expressions but directly receive a `ValueExpression` from the templating language. Both Jakarta Pages 2.1 and Jakarta Faces 1.2 became part of Jakarta EE 5, which was released at the same time.

On June 13, 2006, the MyFaces community announced that the donated project would have its name changed to Trinidad. ADF Faces kept existing at Oracle, though, but was based on Trinidad with some extra features (such as support for Portals, JSR 227, etc.). Just 2 weeks prior to that, on May 31, 2006, ICEsoft announced its free, although still closed source, community edition. A few months later, on November 14, 2006, ICEsoft would fully open source ICEfaces under the MPL license. RichFaces, still closed source at that point and being sold by Exadel, would not stay behind for long though, and some 4 months later, on March 29, 2007, Exadel announced a partnership with Red Hat that made RichFaces available under an open source license and available and supported via its JBoss group.

On to Maturity

On May 22, 2007, the specification work for Jakarta Faces 2.0 began. The scope was hugely ambitious and promised not only to fix many of the issues that people had been complaining about but also to introduce quite a bunch of new features. Mentioned among the many goals in the JSR was a particularly interesting one when looking at the bigger picture—extracting the managed bean facility from Jakarta Faces and making it available for the entire platform.

During the fall of 2007, the community was polled for a name for the Jakarta Faces RI. Four names rose to the top, but as is often the case, none of these names could be approved by Sun's legal department. Eventually, Mojarra was proposed, and perhaps to the surprise of some, this one did pass legal's scrutiny. Ryan Lubke, one of the main Jakarta Faces committers then, made the official announcement on December 5, 2007.

A little under a year later, on October 29, 2008, Çağatay Çivici started a new library, PrimeFaces as a new start after his older YUI4JSF Jakarta Faces component library. The name derives from Çağatay's nickname, which is Optimus Prime, the courageous leader of the heroic autobots in the fictional Transformers universe. Çivici had been involved with Jakarta Faces development for a long time; right after graduation for his bachelor's degree at the university, he started working on a military project that used Jakarta Faces. At the time, this was the closed source Sun version, but Çivici was more interested in MyFaces, since it was already open source. After he provided several patches to this project, he was invited as a committer. Çivici worked for Spring Source for a time while he lived in the UK, working on the Web Flow and Jakarta Faces integration. Ultimately, Spring Source didn't go through with that, after which Çivici worked for Visa, located in Reading, United Kingdom. Many pieces of PrimeFaces were coded by Çivici on his commute from London to Reading.

PrimeFaces was initially based on Jakarta Faces 1.x, but with Jakarta Faces 2.x looming and the project still young, it would soon after switch to Jakarta Faces 2.x.

On July 1, 2009, the long-awaited Jakarta Faces 2.0 finally arrived. Jakarta Faces 2.0 indeed fixed nearly every problem that the industry had with Jakarta Faces; finally, Facelets was included as the default view templating language. Jakarta Pages was effectively deprecated. The state saving concerns that Hookom and Winer brought forward more than 3 years earlier were addressed as well; from then on, Jakarta Faces only saved delta state (state changes), and in restore view, the component tree was reloaded from the template, instead of actually restored.

Another big concern brought forward by the Jakarta Faces community over the years, Jakarta Faces's over-the-top emphasis on postbacks, was addressed too; GET requests became a first-class citizen in Jakarta Faces 2.0. A well-known usability problem with Jakarta Faces, sometimes called "The Trap," was that for a number of operations, the data involved needed to be the same during both the original request and the postback. This is not entirely trivial to guarantee in Jakarta Faces 1.x. Jakarta Faces 2.0 introduced the so-called view scope for this, which elegantly solved the problem. The creation of custom components, yet another problem area of Jakarta Faces 1.x, was made much simpler as well. Jakarta Faces 2.0 also introduced core support for AJAX, modeled after the way Ajax4Jsf worked, a resource API, system events, and quite a few other things.

One of Jakarta Faces 2.0's goals, making its managed bean facility usable outside Jakarta Faces, was implicitly reached by the CDI spec, which was introduced together with Jakarta Faces 2.0 in Jakarta EE 6. The CDI spec itself has a long history too, but one of its defining characteristics is that CDI Beans are strongly based on Jakarta Faces Managed Beans and are essentially a super set of those.

Altogether the impact of all those fixes and new features was such that it split the community essentially in two; those who had used Jakarta Faces 1.x and never looked at it again and those who switched to Jakarta Faces 2.x or, specifically, the ones who started using Jakarta Faces with 2.0 and never saw 1.x. This often led to heated debates, with the 1.x side arguing that Jakarta Faces is horrible and the 2.x side not understanding at all why that would be the case. Even at the time of this writing, which is almost 9 years after Jakarta Faces 2.0 was released, and a longer period than Jakarta Faces 1.x ever existed, these sentiments still remain to some degree.

Despite the many things that Jakarta Faces 2.0 did right, there was one missed opportunity; even though CDI was now available and superseded Jakarta Faces's Managed Beans, Jakarta Faces chose not to deprecate its managed bean facility right away. Even worse, it introduced an annotation-based alternative to the XML-based system Jakarta Faces 1.x used to define managed beans. With CDI already out there having annotations like `javax.enterprise.context.RequestScoped`, simultaneously introducing a `javax.faces.bean.RequestScoped` annotation that did exactly the same thing seems debatable as best. The EG seemed to be aware of this conflict, as a warning was put in place that these new annotations would possibly be superseded by platform functionality before long.

On December 23, Cay Horstmann raised his concerns about this very unwanted situation in an article titled "Is `@javax.faces.bean.ManagedBean` Dead on Arrival?" The response was quite clear; people, including Jakarta EE book writer Antonio Goncalves, asked for this huge mistake that Jakarta Faces 2.0 had made to be corrected as soon as possible and to deprecate `javax.faces.bean.ManagedBean` right away in the upcoming Jakarta Faces 2.1 maintenance release which was called for, among other things, to rectify another mistake (namely, the problem Jakarta Faces 2.0 introduced that in addition to a custom `ResourceResolver`, it was also necessary to provide a custom `ExternalContext`, which was very unclear). Why `javax.faces.bean.ManagedBean` indeed wasn't deprecated in the Jakarta Faces 2.1 MR remains a mystery to this day.

While applications written against the Jakarta Faces 1.x APIs would mostly run unchanged on Jakarta Faces 2.0, or only needed a few small changes, the component libraries had a much harder time. Specifically, the platform-provided AJAX support meant that the existing component libraries would have to forego their own AJAX implementations and rebase on the standard APIs. Clearly, that was no small feat, and it took a long time for component libraries to migrate, with some never really making the switch at all.

Here, PrimeFaces was clearly at an advantage. Being a relatively new library without much legacy, it could switch with much ease, while RichFaces and ICEfaces struggled a lot. For a year, PrimeFaces was really the only component library available for Jakarta Faces 2. PrimeFaces' ascension in popularity started right after Jakarta Faces 2.0 was released, which was also the exact same time that both ICEfaces and RichFaces seemed to become less popular. Although it must be noted that hard statistics are difficult to obtain and contain many facets (downloads, deployments, books, questions asked, available jobs, taking different industries into account, etc.), somewhere around 2012, PrimeFaces had seemingly become the more popular Jakarta Faces component library.

A year earlier, in March 2011, a German software developer working for Samhammer AG named Thomas Andraschko started a project that provided several add-ons for PrimeFaces, called PrimeFaces Extensions. Andraschko would later play a very important role in Jakarta Faces.

In the beginning of that same year, February 19, 2012, Arjan Tijms and Bauke Scholtz (by coincidence also the authors of this book) started the OmniFaces library for Jakarta Faces. The goal of OmniFaces was to be a utility library for Jakarta Faces, essentially what Apache Commons and Google Guava are to Java SE. Tijms and Scholtz had worked on a Jakarta Faces-based website together and found that they both had a collection of private Jakarta Faces utilities that they reused for different projects and also that a great number of similar utilities were essentially rewritten again and again for many Jakarta Faces projects and were partially floating around in places like forum messages and blog posts. OmniFaces was set up in particular not to compete component libraries like PrimeFaces but to work together with those. Hence, visual-oriented components were largely out of scope for OmniFaces.

In 2012, the specification process for Jakarta Faces 2.2 was also in full swing. Jakarta Faces 2.2 was eventually released on May 21, the next year. Jakarta Faces 2.2 specifically came up with a formal version of the alternative mode in which Facelets could operate; instead of putting component tags on a view, plain HTML was put on it, with a special ID linking the tag to a component. Such a mode is generally speaking somewhat less interesting to Jakarta Faces developers but appeals specifically to web designers who can more easily use plain HTML tools for such views. Jakarta Faces 2.2 also introduced a CDI-compatible `@ViewScoped` annotation, which removed one of the last reasons to still use the Jakarta Faces managed bean facility in Jakarta Faces 2.1, namely, that in that version, `@ViewScoped` only worked on those beans. Jakarta Faces 2.2 also introduced two new big features, Faces Flow and Resource Contracts, but these seem to have seen little uptake in practice.

On January 2013, Çağatay Çivici announced that Thomas Andraschko had joined the PrimeFaces team as a committer. Not long after, Andraschko would become the community lead for PrimeFaces and one of the top committers for that project.

Just prior to the start of Jakarta Faces 2.3, on July 20, 2014, RichFaces lead Brian Leathem announced on his blog that RichFaces 5, the next-generation version of RichFaces, would be canceled. Instead, RichFaces would "pursue a path of stability over innovation," which means that JBoss will make RichFaces 4.x compatible with Jakarta Faces 2.2 and port back a few things that were in development for RichFaces 5. While the post was somewhat optimistic, it strongly looked like the writing was on the wall for RichFaces.

On August 26, 2014, the specification work for Jakarta Faces 2.3 started. A new co-spec lead was introduced—Manfred Riem, who up to then had been working mostly on the implementation side of Mojarra, doing such things as migrating hundreds of the tests for which Jakarta Faces is famous away from the ancient and retired Cactus framework to a more modern Maven-based one and making sure the gazillions of open Mojarra issues were reduced to a manageable number. Jakarta Faces 2.3 started off with a perhaps somewhat remarkable message that Oracle had only a few resources available. During the specification process, those few resources dropped to a number that few would have expected—absolutely zero. Basically, after JavaOne 2015, nearly all of the spec leads just vanished, and most specs as a result abruptly ground to a halt. Josh Juneau reported about this in his famous study, "Jakarta EE 8, What Is the Current Status: Case Study for Completed Work Since Late 2015," which undeniably makes it clear by showing graphs of emails, commits, and issues resolved that Oracle had just walked away.

The openness of the Jakarta Faces and its RI Mojarra were fortunately such that the specification work and implementation thereof in Mojarra can largely be carried on by the other EG members, which indeed happens.

Also helping the Jakarta Faces 2.3 efforts and Mojarra was Andraschko, who, in addition to a PrimeFaces committer, had also become a MyFaces committer in July 2015. Andraschko donated a standardized version of the PrimeFaces search expression feature to Jakarta Faces 2.3 and Mojarra.

Meanwhile on February 12, 2016, Red Hat announced that RichFaces would be end of lived (EOL) later that year, namely, in June 2016. One of the most popular Jakarta Faces component libraries at some point, often named something like "One of the big three," effectively was no more. On June 20, 2016, the last real commit to the project was done, "RF-14279: update JSDoc." Two days later, Red Hat released RichFaces 4.5.17, and the GitHub repos were put into archived (read only) mode. Brian Leathem, who is still a Jakarta Faces 2.3 EG member, announced a few days later on February 18 that he would no longer be doing any Jakarta Faces-related work.

Rejuvenation

In late 2016, the Jakarta Faces spec leads briefly returned, but with the message that the spec must be completed in only a few weeks, so the (somewhat) lengthy finalization process could start. On March 28, 2017, Jakarta Faces 2.3 was then eventually released, bringing with it the start of replacing Jakarta Faces native artifacts with CDI versions and finally something which should have happened years ago: the deprecation of the Jakarta Faces managed bean facility in favor of using CDI beans. Other features are support for WebSocket using the Jakarta EE WebSocket APIs donated by OmniFaces, the introspection of available view resources in the system, and the abovementioned search expression framework donated by PrimeFaces.

Following the somewhat turbulent development of the Jakarta Faces 2.3 spec is the even more turbulent announcement by Oracle in 2017 that Jakarta EE, thus including Jakarta Faces, would be transferred to the Eclipse Foundation. Oracle would stop leading the specs it owned before, which again includes Jakarta Faces. This would mean that Mojarra would be re-licensed, and Jakarta Faces would be evolved by a new process with different leads.

The transition from Oracle to Eclipse was done in several steps. A new top-level project called Eclipse Enterprise for Java (EE4J) was created, and a snapshot of the Jakarta Faces 2.3 branch was scrubbed (legally speaking) and moved to a new GitHub repository at `http://github.com/eclipse-ee4j`. Only Jakarta Faces 2.3 (and associated Mojarra 2.3) was moved, meaning Jakarta Faces 2.2 and earlier remained at their Oracle locations. This specifically meant the commit history wasn't transferred. After this first step of the transfer, a new release of the API was done in January 2019 as *jakarta.faces:jakarta.faces-api:2.3.1*, which was mostly identical to the existing *javax.faces:javax.faces-api:2.3*. The license of both the API and the Mojarra implementation was changed to the Eclipse Public License 2.0. Arjan Tijms became the next project lead. Previous spec leads Ed Burns and Manfred Riem shortly after left Oracle to work at Microsoft. Riem, however, would stay involved with Jakarta Faces.

The next step in the transfer process involved creating a new certification process and a new specification license. The JCP process was replaced by the Jakarta EE Specification Process (JESP), and the new specification license became the Eclipse Foundation Technology Compatibility Kit License (EFTL).

Early in 2019, Tijms went to the Javaland conference in Germany, where he met up with Ivar Grimstad, the Jakarta EE Developer Advocate at the Eclipse Foundation. In the warm sun of the German afternoon, the two discussed a new naming scheme for the Jakarta EE specifications. New names were required, as the deal between Oracle and the Eclipse Foundation included an agreement to not use a number of names previously used by Oracle, such as Jakarta EE, Jakarta Faces, JavaServer Faces, and Jakarta Pages. Tijms proposed a simpler naming scheme, essentially abolishing as much of the various "filler words" (such as "service," "api," and "architecture") in the names, making cryptic abbreviations (like, indeed, Jakarta Faces) necessary. Ivar agreed with this, and in the following months, this naming scheme was introduced; code, documentations, and specifications were painstakingly combed through to find and replace occurrences of the old names with the new names. For Jakarta Faces, the name became "Faces," a logical choice, since from day one the package name had actually been *javax.faces.**, not *javax.jsf.**. The artifact id also had been "faces" for a long time.

Later that year, in September 2019, the Faces 2.3.2 API was released as part of Jakarta EE 8. This API was again largely the same as the Jakarta Faces 2.3.1 API but now had all terms replaced by the new ones, and it was certified using the new JESP process.

That same month, during the JakartaOne Livestream 2019, Tijms, from his hotel in Jakarta, Indonesia, presented some of the ideas for the next version of Faces, in a presentation called "What's Coming to Jakarta Faces?" In it, Tijms mentioned that central for that release would be the removal of deprecated items, such as the Faces native expression language that was deprecated in 2005, Jakarta Server Pages support that was deprecated in 2009, the Faces native Managed Beans (deprecated in 2016), and more. With regard to features, Tijms mentioned extensionless views by default and a new life cycle phase to make things for which the view action is now used on an empty page easier to use.

After this second step was completed, the third step started. This involved changing the namespace of all Jakarta EE APIs, which of course included Faces, from javax.* to jakarta.*. This was needed since per the JCP rules, packages are only allowed to be modified per the JCP process. As Jakarta EE was no longer following that process, the package had to be changed. Tijms and other committers worked hard the following months to change the package names in the source code and documentation, which eventually resulted in the release of Faces 3.0 in October 2020 as part of Jakarta EE 9. Due to time constraints, the original plan of certifying Jakarta EE 9 (and therefore Faces 3.0) for JDK 11 compatibility had to be dropped.

Shortly after the release of Faces 3.0, Tijms kicks off the work for Faces 4.0 with a mail to the faces-dev list titled “Starting Jakarta Faces 4.0.” The mail reiterates some of the plans from the earlier presentation but adds several API enhancements proposed earlier by Thomas Andraschko.

On October 23, 2020, Arjan Tijms sets the version of the Mojarra branch to 4.0.0-SNAPSHOT, marking the start of Faces 4.0. A few days later, on October 27, 2020, a historical commit is done by Tijms: “Remove Jakarta Pages support.” That one technical decision, which in 2003 was strongly criticized by Hans Bergsten and that indeed as Bergsten predicted would bring so much pain to Faces, is finally undone. Again a few days later, on November 1, 2020, the native managed bean system of Faces also bites the dust. CDI beans from that moment on are the only designated bean type for Faces.

Early 2021, Tijms, Andraschko, and Scholtz discuss how to go forward with the proposed changes, and the next months see a flurry of new issues being created and commits done in both the Mojarra and MyFaces projects to implement several of those proposed new features.

Simultaneously, the Jakarta EE TCK is updated to support JDK 11, which eventually results in a Jakarta EE 9.1 release in May 2021. As both the Faces API and the Mojarra 3.0 implementation are already JDK 11 compatible, no new releases were required for those.

The following table shows the timeline and the differences between the various versions released:

Date	Maven coordinates	Build from	Certified	Javadoc	Java package	TCK level	API level
2017, Mar	javax. faces:javax. faces-api:2.3	github/ javaee	Jakarta EE 8	Jakarta EE terms	javax. faces	JDK 8	JDK 8
2019, Jan	**jakarta**. faces:**jakarta**. faces-api:2.3.1	github/**eclipse-ee4j**	Jakarta EE 8	Jakarta EE terms	javax. faces	JDK 8	JDK 8
2019, Sep	**jakarta**. faces:**jakarta**. faces-api:2.3.2	github/**eclipse-ee4j**	**Jakarta** EE 8	**Jakarta EE** terms	javax. faces	JDK 8	JDK 8

Date	Maven coordinates	Build from	Certified	Javadoc	Java package	TCK level	API level
2020, Oct	***jakarta**. faces:**jakarta**. faces-api:3.0*	*github/**eclipse-ee4j***	***Jakarta** EE 9*	***Jakarta EE** terms*	***Jakarta**. faces*	*JDK8*	*JDK 8*
2021, May	***jakarta**. faces:**jakarta**. faces-api:3.0*	*github/**eclipse-ee4j***	***Jakarta** EE 9.1*	***Jakarta EE** terms*	***Jakarta**. faces*	*JDK**11***	*JDK 8*

On August 21, 2021, Çağatay Çivici presents the latest state of Faces in a presentation for the Turkish edition of the JakartaOne Livestream. His presentation is aptly titled "Jakarta Faces 4.0." At the moment of writing, the release review for Faces 4.0 is planned to start at around May 15, 2022.

CHAPTER 2

From Zero to Hello World

In this chapter, you will learn how to set up a Jakarta Faces development environment with the Eclipse IDE (integrated development environment), the Maven dependency management system, the WildFly application server and the H2 database from scratch.

Installing Java SE JDK

You probably already know that Java SE is available as JRE for end users and as JDK for software developers. Eclipse itself does not strictly require a JDK as Eclipse has its own compiler. Jakarta Faces being a software library does not require a JDK to run either. WildFly, however, does require a JDK to run, primarily in order to be able to compile Jakarta Pages files, even though Jakarta Pages (formerly known as Jakarta Pages, Java Server Pages) has, since Jakarta Faces 2.0, been deprecated as view technology for Jakarta Faces.

Therefore, you need to make sure that you already have a JDK installed as per Oracle's instructions. At the time of writing, the latest available Java SE version is 17, but as Jakarta EE 10 was designed for Java SE 11, you could get away with a minimum version of Java SE 11. Installation instructions depend on the platform being used (Windows, Linux, or MacOS). You can find detailed Java SE 17 JDK installation instructions here: `https://docs.oracle.com/en/java/javase/17/install/overview-jdk-installation.html`

The most important parts are that the `PATH` environment variable must cover the `/bin` folder containing the Java executables (e.g., "`/path/to/jdk/bin`") and that the `JAVA_HOME` environment variable is set to the JDK root folder (e.g., "`/path/to/jdk`"). This is not strictly required by Jakarta Faces itself, but Eclipse and WildFly need this. Eclipse as being an integrated development environment will need the `PATH` in order to find the Java executables. WildFly as being a Jakarta EE application server will need the `JAVA_HOME` in order to find the JDK tools.

B. Scholtz and A. Tijms, *The Definitive Guide to Jakarta Faces in Jakarta EE 10*,
https://doi.org/10.1007/978-1-4842-7310-4_2

What About Jakarta EE?

Jakarta Faces itself is part of Jakarta EE. Jakarta EE is basically an abstract specification of which the so-called application servers represent the concrete implementations. Examples of those application servers are Eclipse GlassFish, Red Hat WildFly, Apache TomEE, and IBM WebSphere Liberty. You can find them all at the Jakarta EE Compatible Products page: `https://jakarta.ee/compatibility/`. It is exactly those application servers that actually provide among others Jakarta Faces, Jakarta Expression Language (EL), Jakarta Tags (formerly known as Jakarta Tags), Jakarta Contexts and Dependency Injection (CDI), Jakarta Enterprise Beans (formerly known as Jakarta Enterprise Beans, Enterprise Java Beans), Jakarta Persistence, Jakarta Servlet, Jakarta WebSocket, Jakarta JSON Processing, and many more APIs (application programming interfaces) of the box.

There also exist so-called servlet containers that provide basically *only* the Jakarta Servlet, Jakarta Pages, Jakarta Expression Language, Jakarta WebSocket, and Jakarta Security out of the box, such as Apache Tomcat and Eclipse Jetty. Therefore, it would require some additional work to manually install and configure, among others, Jakarta Faces, Jakarta Tags, Jakarta Contexts and Dependency Injection, Jakarta Enterprise Beans, Jakarta Persistence, Jakarta JSON Processing, and/or many other missing APIs from the Jakarta EE platform on such a servlet container. It is not even trivial in the case of Jakarta Enterprise Beans as it requires modifying the servlet container's internals. That is, by the way, exactly why Apache TomEE exists. It's a Jakarta EE application server built on top of the bare-bones Tomcat servlet container engine.

So you do not need to download and install "Jakarta EE" as a whole. Basically downloading any Jakarta EE application server to your choice is sufficient. We're going to pick Red Hat WildFly for this book.

Installing WildFly

WildFly is an open source Jakarta EE application server from Red Hat. You can download it from `https://wildfly.org/downloads`. Make sure you choose the "Jakarta EE 10 Full & Web Distribution" and not, for example, the "Servlet-Only Distribution" or "Preview Distribution," as they have other purposes. Installing is basically a matter of unzipping the downloaded file and putting it somewhere in your home folder. We'll leave it there until we have Eclipse up and running so that we can then integrate WildFly in Eclipse and let Eclipse manage the WildFly application server.

Installing Eclipse

Eclipse is an open source IDE written in Java. You can download it from `https://eclipse.org`. It is basically like notepad but then with thousands if not millions of extra features, such as automatically compiling class files, building a WAR (Web Application Archive) file with them, and deploying it to an application server without the need to manually fiddle around with `javac` and others in a command console.

Eclipse is available in a lot of flavors, even for C/C++ and PHP. As we're going to develop with Jakarta EE (Enterprise Edition), we need the one saying "Eclipse IDE for **Enterprise** Java and Web developers," importantly the one with "Enterprise" in its name. The one without it doesn't contain the mandatory plug-ins for developing Jakarta EE web applications. Here is the main download page: `https://www.eclipse.org/downloads/packages/`. Also here, installing is basically a matter of unzipping the downloaded file and putting it somewhere in your home folder.

In Windows and Linux, you'll find the `eclipse.ini` configuration file in the unzipped folder. In MacOS, this configuration file is located in `Eclipse.app/Contents/Eclipse`. Open this file for editing. We want to increase the allocated memory for Eclipse. At the bottom of `eclipse.ini`, you'll find the following lines:

```
-Xms256m
-Xmx2048m
```

This sets, respectively, the initial and maximum memory size pool which Eclipse may use. This is a bit too low when you want to develop a bit of a decent Jakarta EE application. Let's at least double both the values.

```
-Xms512m
-Xmx4g
```

Watch out that you don't declare more than the available physical memory in the `Xmx` setting. When the actual memory usage exceeds the available physical memory, then it will continue into virtual memory, usually in flavor of a swap file on disk. This will greatly decrease performance and result in major hiccups and slowdowns.

Now you can start Eclipse by executing the `eclipse` executable in the unzipped folder. You will be asked to select a directory as workspace. This is the directory where Eclipse will save all workspace projects and metadata.