IN THE UNITED STATES DISTRICT COURT

FOR THE WESTERN DISTRICT OF WISCONSIN

- -

FREYBURGER LLC,

Plaintiff,

v.

MICROSOFT CORPORATION,

Defendant.

OPINION and ORDER

09-cv-104-bbc

- -

This case for patent infringement is before the court for construction of several terms in U.S. Patent No. 6,405,368, a patent related to computer programming owned by plaintiff Freyburger LLC that discloses "a method and apparatus for separate compilation of templates." With respect to almost every term at issue, the parties' proposed constructions are ships passing in the night. Despite the many differences, the parties generally focused on one or two narrow disputes both in their briefs and at the claims construction hearing. I will follow the parties' lead. In this opinion, I will address only the disputes articulated by the parties. In those situations in which the parties did not adequately justify or explain their chosen language, I have disregarded their proposals.

The absence of a construction in this opinion should not be read by the parties to

mean that no construction is necessary. District courts have an obligation to construe terms when it is necessary to resolve a genuine and material legal dispute between the parties. O2 Micro Intern. Ltd. v. Beyond Innovation Technology Co., Ltd., 521 F.3d 1351, 1361-62 (Fed. Cir. 2008). If a party shows at summary judgment or at trial that further clarification is needed to resolve a material dispute, the court will provide it. However, courts have no obligation to provide constructions simply because the parties request them; the parties must demonstrate that the construction is both necessary and correct. Id.; see also E-Pass Techs., Inc. v. 3Com Corp., 473 F.3d 1213, 1219 (Fed. Cir. 2007) ("[A]ny articulated definition of a claim term ultimately must relate to the infringement questions that it is intended to answer."). In many instances, the parties failed to show that a construction was needed, or, if it was needed, that the proposed construction was supported by the patent or the extrinsic evidence.

In light of this problem, there are only two partial constructions that I am able to provide at this stage. I conclude that a template can be "filled in by a compiler during compilation" and that a template definition "uses the keyword 'template.'" Any further constructions must await summary judgment or trial.

OPINION

A. Asserted Claims

Claims 3 and 5 are at issue in this case:

> 3. A method of creating an executable program from source programs having templates comprising the steps of:
>
> a) providing a first source file having a template definition;
>
> b) compiling the first source file into a first object file;
>
> c) providing a second source file having a template instantiation;
>
> d) compiling the second source file into a second object file, the step of compiling the second source file into a second object file having as inputs the second source file and a translation of the template definition.
>
> 5. A method of creating an executable program from source programs having templates comprising the steps of:
>
> a) providing a first source file having a template definition;
>
> b) compiling the first source file into a first object file;
>
> c) providing a second source file having a template instantiation;
>
> d) compiling the second source file into a second object file, the step of compiling the second source file into a second object file having as inputs the second source file and the template definition from the first object file.

B. Template

Plaintiff's Proposed Construction: a set of parameterized classes, functions, or variables

Defendant's Proposed Construction: a source code substitution feature that defines a set of parameterized types or functions using placeholders that can be filled in by a source code compiler during compilation of a source file

The parties devoted much of their efforts in the briefs and at the claim construction hearing to this term. Interestingly, both sides agree that a "template" is a familiar term in the art, Dft.'s Br., at 14-15, dkt. #34; Plt.'s Br., at 6, dkt. #32, but neither side did a very good job of explaining the concept. Plaintiff says that this term and several others need no explanation because they are "ordinary computer programming terms" and are "understandable to a jury as written," Plt.'s Br., at 1, dkt. #32, but this assumes incorrectly that the jury will consist of computer programmers. Defendant says that it is difficult to describe templates concisely because "[m]ost programming books . . . devote entire chapters to explaining them," Dft.'s Br., at 15, dkt. #34, so it focused on the aspects of the concept that it believes are important to this case.

At the claim construction hearing, the parties agreed that a template involves "parameterized types" in the sense that it uses "placeholders" that can be filled in later. However, defendant argues that a "template" is a very specific kind of "parameterized type" that is not reflected in plaintiff's proposed construction.

As defendant frames it, the issue is whether a template is used only during

compilation (that is, during the conversion of the programming language into the binary code of zeros and ones that can be read by the computer) or whether it is used during runtime as well (that is, when the program is being executed). However, that is not an issue that can be resolved in this opinion.

The extrinsic evidence supports a reading that templates are "instantiated" by the compiler during compilation. (The parties agree that "instantiation" is the process by which templates are filled in with information.) The references cited by defendant are uniform in including this reading in their description of templates. E.g., Nishant Sivakumar, C++/CLI in Action 124 (2007) (included as exh. C to dkt. #34) ("templates" are "[i]nstantiated during compilation by the C++ compiler"); Stephen R.G. Fraser, Pro Visual C++/CLI and the .NET 2.0 Platform 164 (2006) (included as exh. D) ("One major difference between generics and templates is when they are instantiated. For templates, it happens at compile time, whereas for generics, it happens at runtime."); Bruce Eckel, Thinking in C++ 582 (1995) (included as exh. H) ("When you use a template, the parameter is substituted by the compiler.").

Nothing in the patent undermines this understanding. Plaintiff is quick to point out that all of defendant's references discuss "templates" in the context of the programming language C++ and that the '368 patent is not limited to that programming language. Both of these observations are true, but they do not provide support for a view that the references

cited by defendant are inaccurate reflections of the term in the context of the patent. The specification makes it clear that the concept of templates in C++ is the same concept used by the patent: "The ANSI/ISO working group X3J16/WG21 has standardized a new programming language, C++, based upon C. One of the many features included in the language is 'templates.'" '368 pat., col. 1, lns. 11-14. The specification goes on to discuss templates in the context of the invention without any hint that the invention employs a different understanding of templates. It states explicitly: "Though this described embodiment of the invention is described in terms of C++, it can be generally applied to any programming language which has a similar notion of templates." '368 pat., col. 2, lns. 64-67. Thus, the invention may apply to languages other than C++, but the "notion of templates" in the patent comes from that programming language.

Plaintiff raises another objection to defendant's references, which is that many of them postdate the patent. This is a red herring. Plaintiff does not cite any evidence to suggest that the meaning of "template" in the context of the C++ language has changed since the date of the invention.

In any event, the specification includes several passages in which the inventor states or assumes that the template is instantiated by the compiler during compilation. E.g., '368 pat., col. 1, lns. 18-21 ("When a compiler encounters . . . a reference [to a template], the definition of the referenced template instance is generated from the template definition

through a process called instantiation."); id. at col. 4, lns. 13-14 ("Whenever the compiler instantiates a template . . ."). This confirms the understanding of the term found in the extrinsic evidence.

It is not clear how far this conclusion gets defendant because its proposed construction goes beyond what its evidence supports. Defendant does not ask for a construction stating that a template is "filled in during compilation." Rather, its proposed construction reads: a template is "filled in by a source code compiler during compilation of a source file." As plaintiff points out, the term "source code compiler" does not appear in the patent or in any of the extrinsic evidence cited by either party.

It may be that inclusion of the modifier "source code" in the proposed construction is redundant. After all, even plaintiff seems to believe that compilation inherently involves the translation of source code into machine code. Plt.'s Br., dkt. #32, at 5 ("Compilation is thus the process of translating source code . . .to binary code.") However, because this is not an issue developed by the parties, I decline to resolve it at this time.

Unfortunately, defendant fails to develop any meaningful argument with respect to the rest of its proposed construction, so I do not adopt it. The same is true of plaintiff's proposed construction, "a set of parameterized classes, functions, or variables." Plaintiff makes little effort to explain how its construction would be helpful in resolving any issues in the case or improving a jury's (or the court's) understanding of the term. In fact, the

construction provides little indication that it is related to programming at all.

Plaintiff's justification for its construction is a line from the background section of the specification: "A template allows the definition of a set of classes, functions or variables, parameterized by a set of template parameters." '368 pat., col. 1, lns 14-16. This passage may suggest that all or some of plaintiff's proposal should be *part* of a construction of "template," but the passage itself makes it clear that it is not intended as a definition. It states what a template "allows;" not what it *is*. The extrinsic evidence plaintiff cites is the same. Plaintiff never addressed this problem in its briefs or at the hearing. Even if I agreed with plaintiff that the construction should be lifted from the specification, I still could not adopt plaintiff's proposal because it omits the end of the sentence ("by a set of template parameters"). Again, plaintiff fails to explain the omission.

The parties have failed to suggest a definition of "template" that I can adopt. Thus, the only issue regarding this term that may be resolved at this stage of the proceedings is that template can be "filled in by a compiler during compilation."

C. Template Definition

Plaintiff's Proposed Construction: definition of a set of parameterized classes, functions, or variables

Defendant's Proposed Construction: source code that defines a set of classes or functions using the keyword "template"

As with "template," the parties have provided little help. Plaintiff's proposed construction simply adds the phrase "definition of" in front of the same proposed construction for "template" that I have not found persuasive. Defendant takes the opposite tack; its proposed construction for "template definition" bears virtually no resemblance to its proposed construction for "template." This is a curious result that defendant explains only by saying that a "template definition is a coined term of art that has a precise meaning in the field." Dft.'s Br., at 21, dkt. #52. This is not a helpful explanation.

Again, it is unnecessary to provide a complete construction of the term at this stage; it is only necessary to resolve the parties' articulated disputes. This term has two, whether the template definition must take the form of source code and whether it uses the keyword "template." Defendant says that both limitations apply; plaintiff says neither do.

Although defendant has failed to show that a template definition must be source code, I agree that the keyword "template" should be included in the construction. With respect to the source code limitation, plaintiff cites several passages from the specification showing that the template definition is not always source code. For example, in a discussion of the template definition, the specification states that the invention can "actually produce machine code for the parts of the template definition which do not depend upon the template parameters." '368 pat., col. 5, lns. 52-54. (The parties agree that machine code is another way of describing binary code.) Defendant does not even try to address this or similar

passages.

In its opening brief, defendant cites the claim language itself as proof that the template definition is source code: "providing a first source file having a template definition." '368 pat., col. 6 lns. 36-37 and 50-51. Defendant does not develop this argument, but it seems to rest on the assumption that a template definition must be source code if it is included in a source file. The assumption seems questionable in light of another passage cited by plaintiff: "[t]he source file includes both source code and definitions of templates." '368 pat., col. 1, lns. 26-27. If the template definition were always source code, one would think that the inventor would not discuss the two things as distinct. (The cited passage comes from a discussion of the background of the invention rather than the invention itself, but neither party suggests any reason to believe that the claimed invention is any different in this respect.) Further, even if I accepted defendant's assumption that the template definition's location in a particular file governs the form the definition takes, this would not help defendant. Claim 5 says that the template definition comes "from the first object file." Under defendant's view, this would mean that the template definition would have to be object code (another synonym for binary code or machine code) in that instance.

With respect to the proposed limitation of "using the keyword template," defendant again cites multiple pieces of extrinsic evidence that describe a "template definition" this way. E.g., Ray Lischner, Exploring C++ 394 (2009) ("The template key word means that

what follows is a template."); Bruce Eckel, Thinking in C++ 583 (1995) (stating that "[t]he template keyword" is part of "template syntax"). Plaintiff seems to agree that the keyword "template" is part of the template definition of the C++ language, but plaintiff repeats its argument that the '368 patent is not limited to that language. Again, this observation is accurate but unpersuasive. Whatever language is at issue, the specification is clear that the invention uses the notion of templates from the C++ language. The only time the patent provides an example of a template definition, it uses the "template" keyword:

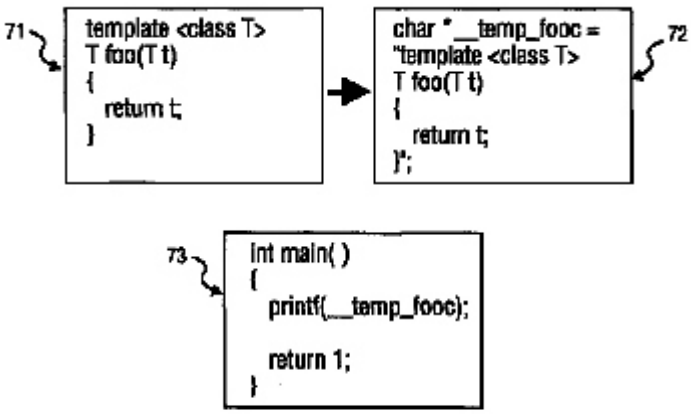

FIG. 4

Citing this embodiment is not importing limitations from the embodiments into the claims

as plaintiff suggests; it is simply further confirmation that the inventor has incorporated the C++ notion of templates into the '368 patent.

Accordingly, I conclude that a template definition "uses the keyword 'template.'"

D. Template Instantiation

Plaintiff's Proposed Construction: a reference to a template that specifies the template parameter(s)

Defendant's Proposed Construction: a reference in source code that specifies the template parameter(s) to enable the process by which the source code compiler converts a template definition into a specific class or function

The parties agree that the word "instantiation" is used two different ways throughout the '368 patent. First, it is the "filling in" process during compilation, in which the "blanks" in the template are replaced with integers, character strings or another data type. Dft.'s Br., at 3, dkt. #34 ("When the template is 'instantiated' with an integer (i.e., has its blanks filled in with an integer during compilation), the template source code is transformed into a function to sort integers.") The parties raise no disputes regarding this use of the term, which plaintiff describes as the "verb" form of the term. Rather, the parties dispute the "noun" form.

Unfortunately, in their briefs, the parties largely talked past each other in addressing this term, in part because defendant changed its proposed construction between its opening

brief and response brief. The parties did not use the claims construction hearing to remedy this problem; both sides ignored the term. Because the parties have not joined issue, any construction of this term will have to wait for summary judgment or trial.

E. Source File

Plaintiff's Proposed Construction: computer file containing human-readable code in a high-level or assembly language

Defendant's Proposed Construction: an uncompiled computer file written by a programmer in a high level computer programming language

In its opening brief, defendant frames the question for this term as whether source files contain source code, even though the words "source code" do not appear in its construction. This is another example of a party failing to match up its arguments with its proposed construction. The articulated disputes between the parties are three: (1) whether a source file must be "written by a programmer"; (2) whether a source file may be written in an "assembly" language; and (3) whether source files must be "uncompiled."

Unfortunately, the parties fail to explain how any of these limitations is important to the issues in the case. Defendant does not describe the purpose of the phrases "written by a programmer" and "uncompiled," except to state in the most general terms that they will show that defendant does not infringe plaintiff's patent. That is not sufficient. The same

is true for plaintiff's proposed addition of "assembly language." Defendant argues that the phrase is irrelevant to the case, a contention that plaintiff does not deny. Accordingly, I decline to construe the term "source file" at this time.

F. Object File

Plaintiff's Proposed Construction: computer file generated by a compiler (or an assembler) that was translated from source code and that may be linked with one or more compiled modules or data

Defendant's Proposed Construction: a file translated from source code and containing the output of a compiler that is formatted to be linked with other object files into an executable form

The parties focus on the last phrase in defendant's proposed construction: "formatted to be linked with other object files into an executable form." The parties agree that the inventor was not using a specialized understanding of "object file," so defendant relies on the IEEE Standard Dictionary of Electrical and Electronics Terms (1996). Verizon Services Corp. v. Vonage Holdings Corp., 503 F.3d 1295, 1304 (Fed. Cir. 2007) (relying on dictionary to define term when specification did not provide definition). That dictionary defines the term in part as "a regular file containing the output of a compiler, formatted as input to a linkage editor for linking with other object files into an executable form." Dkt. #34, exh. K. Defendant modifies this definition without explanation, deleting the phrase "as input to a linkage editor."

The bigger problem with defendant's proposed construction is that the word "formatted" is another ambiguous term. Defendant does not say so explicitly, but it seems to read the phrase "formatted to be linked with other object files" to mean "*must* be linked with other object files" or "unable to link with anything besides other object files." However, defendant fails to support this reading with any evidence.

Defendant cites a number of passages from the specification, but none of them supports its reading. These passages provide examples in which object files *are* linked with other object files, but none of them *requires* object files to be linked with other object files or implies that object files cannot be linked to other types of files. E.g., '368 pat., col. 1, lns. 32-33 ("A linker process 134 combines multiple object files into a single executable program 144."); id. at col 1., lns. 66-67 (invention "link[s] the first and second object files into an executable program"). Thus, defendant has failed to show that its proposed construction is appropriate or even that its proposed construction means what defendant says it means.

In the materials that plaintiff submitted at the hearing, plaintiff changes the phrase "was translated from source code" to "contains a direct or indirect translation of source code." Plaintiff did not articulate an argument in support of this change in its briefs or at the hearing. In any event, I would not adopt plaintiff's proposed construction because "[t]he terms 'direct' and 'indirect' are themselves ambiguous . . . invit[ing] further debate at summary judgment regarding the meaning of those terms." Extreme Networks, Inc. v.

Enterasys Networks, Inc., 07-C-229-C, 2007 WL 5601497, *12 (W.D. Wis. Nov. 21, 2007). Whatever limitation plaintiff is trying to advance with the word "indirect," it will have to develop at summary judgment or trial.

G. Step of Compiling

the step of compiling the second source file into a second object file having as inputs the second source file and *the template definition from the first object file* (claim 5)

Plaintiff's Proposed Construction: no construction necessary

Defendant's Proposed Construction: using the template definition stored in the first object file to instantiate the template during the compiling of the second source file into a second object file

the step of compiling the second source file into a second object file having as inputs the second source file and *a translation of the template definition* (claim 3)

Plaintiff's Proposed Construction: no construction necessary

Defendant's Proposed Construction: using the translation of the template definition to instantiate the template during the compiling of the second source file into a second object file

The only difference between these two terms is the italicized phrase in each. Defendant's proposed constructions raise red flags immediately because they seem to be such a dramatic departure from the claim language and because they require rearranging words, inserting one phrase and omitting others.

Distilled, the dispute seems to be whether these steps must include the template instantiation process. Defendant has an uphill battle proving this limitation because it is nowhere to be found in the language of the claim. Defendant's primary argument is that the invention cannot perform one of its stated purposes unless template instantiation occurs during this step. Plaintiff disagrees, arguing that the instantiation process is not required in each instance.

Whether defendant or plaintiff is correct about the operation of the invention is an issue I need not resolve at this time. Even if defendant is correct, that is an issue of validity, not claims construction. Defendant cites no authority for the proposition that language with no grounding in the claims may be imported because the invention will not work properly without a particular limitation. If defendant can prove that plaintiff's invention is inoperable as written, it may raise that issue at summary judgment.

H. Translation of the Template Definition

Plaintiff's Proposed Construction: a translation of the template definition (i.e., conversion from its original language)

Defendant's Proposed Construction: a representation of the template definition stored in an object file

The parties identify two issues: (1) whether a translation is a representation; and (2)

whether the translation is stored in an object file. With respect to the first dispute, defendant points to no evidence in the specification or elsewhere that would suggest that "representation" and "translation" are interchangeable. At best, defendant could argue that a translation is one kind of representation.

With respect to the second dispute, the language of the claims themselves defeats defendant's argument that the "translation of the template definition" must be stored in an object file. As plaintiff points out, *Claim 5* does include such a limitation; if claim 3 were meant to have one as well, the inventor would have made that explicit. Miken Composites, L.L.C. v. Wilson Sporting Goods Co.,515 F.3d 1331, 1337 (Fed. Cir. 2008) (internal quotations omitted) ("Had the patentee, who was responsible for drafting and prosecuting the patent, intended something different, it could have prevented this result through clearer drafting."); Inpro II Licensing, S.A.R.L. v. T-Mobile USA, Inc., 450 F.3d 1350, 1354 (Fed. Cir. 2006) ("[T]he doctrine of claim differentiation means that different claims are presumed to be of different scope."). Defendant cites a passage from the prosecution history that it says supports its position, but that passage is not very helpful. It includes the phrase "object file" in parentheses after the phrase "translation of the template definition," dkt. #34, exh. L, but defendant provides no reason why the phrase is meant to be a limitation on the invention as a whole rather than a description of the embodiment in claim 5.

I. Conclusion

Claim construction is never an easy process, but it was made much more difficult in this case by the lack of context for the parties' dispute. It has become clear to me that If I am going to be able to decide the many disputes remaining in the case, I will need some sort of introduction to the basic concepts of computer programming that are relevant to this case.

It would be helpful to have a two to three hour tutorial presented by any expert counsel can agree on. I contemplate a format that would consist primarily of a lecture by the expert, with questions from counsel as well as the court, although I would expect counsel to limit their questions to basic matters that would provide context to issues in dispute and not to questions bearing directly on those disputed issues. Of course, the expert would be independent of both parties.

The morning of October 23, 2009 (a Friday) is open on my calendar, if counsel can make the necessary arrangements by then. If that is not convenient, or if counsel have additional questions, they should advise the clerk of court and I will set up a telephone scheduling conference hearing with them.

ORDER

IT IS ORDERED that

1. The claims in U.S. Patent No. 6,405,368 are construed as follows:

- a template can be “filled in by a compiler during compilation”; and
- a template definition “uses the keyword ‘template.’”

2. The clerk of court is directed to set a hearing in this case for October 23, 2009, at 8:30 a.m. in which an expert chosen by both sides shall give a presentation on the technology relevant to the issues in the case. If the parties have any scheduling conflicts or they cannot reach an agreement on an appropriate expert, they should advise the clerk of court and I will set up a telephone hearing to resolve their conflicts.

Entered this 21st day of September, 2009.

BY THE COURT:
/s/
BARBARA B. CRABB
District Judge