WebAssembly Spec Addendum: Legacy Exception Handling

WebAssembly Community Group
Andreas Rossberg (editor)

Nov 03, 2025

Contents

1 Introduction

This document describes an extension of the official WebAssembly standard developed by its W3C Community Group[1] with additional instructions for exception handling. These instructions were never standardized and are deprecated, but they may still be available in some engines, especially in web browsers.

2 Structure

2.1 Instructions

---

[1] https://www.w3.org/community/webassembly/

Control Instructions

The set of recognised instructions is extended with the following:

$$
\begin{array}{rcl}
instr & ::= & \ldots \\
& | & \text{try } \mathit{blocktype}\ instr^*\ (\text{catch } \mathit{tagidx}\ instr^*)^*\ (\text{catch\_all } instr^*)^?\ \text{end} \\
& | & \text{try } \mathit{blocktype}\ instr^*\ \text{delegate } \mathit{labelidx} \\
& | & \text{rethrow } \mathit{labelidx}
\end{array}
$$

The instructions try and rethrow, are concerned with exceptions. The try instruction installs an exception handler, and may either handle exceptions in the case of catch and catch_all, or rethrow them in an outer block in the case of delegate.

The rethrow instruction is only allowed inside a catch or catch_all clause and allows rethrowing the caught exception by lexically referring to a the corresponding try.

When try-delegate handles an exception, it also behaves similar to a forward jump, effectively rethrowing the caught exception right before the matching end.

3 Validation

3.1 Conventions

Contexts

The context is enriched with an additional flag on label types:

$$
\begin{array}{rcl}
labeltype & ::= & \text{catch}^?\ resulttype \\
C & ::= & \{\ldots, \text{labels } labeltype^*, \ldots\}
\end{array}
$$

Existing typing rules are adjusted as follows:

- All rules that extend the context with new labels use an absent catch flag.

- All rules that inspect the context for a label ignore the presence of a catch flag.

> **Note**
>
> This flag is used to distinguish labels bound by catch clauses, which can be targeted by rethrow.

3.2 Instructions

Control Instructions

try $\mathit{blocktype}\ instr_1^*$ (catch $x\ instr_2^*$)* (catch_all $instr_3^*$)$^?$ end

- The block type must be valid as some function type $[t_1^*] \to [t_2^*]$.
- Let C' be the same context as C, but with the label type $[t_2^*]$ prepended to the labels vector.
- Under context C', the instruction sequence $instr_1^*$ must be valid with type $[t_1^*] \to [t_2^*]$.
- Let C'' be the same context as C, but with the label type catch $[t_2^*]$ prepended to the labels vector.
- For every x_i and $instr_{2i}^*$ in (catch $x\ instr_2^*$)*:
    - The tag $C.\text{tags}[x_i]$ must be defined in the context C.
    - Let $[t_{3i}^*] \to [t_{4i}^*]$ be the tag type $C.\text{tags}[x_i]$.
    - The result type $[t_{4i}^*]$ must be empty.
    - Under context C'', the instruction sequence $instr_{2i}^*$ must be valid with type $[t_{3i}^*] \to [t_2^*]$.
- If (catch_all $instr_3^*$)$^?$ is not empty, then:

- Under context C'', the instruction sequence $instr_3^*$ must be valid with type $[] \to [t_2^*]$.

- Then the compound instruction is valid with type $[t_1^*] \to [t_2^*]$.

$$\frac{\begin{array}{c} C \vdash blocktype : [t_1^*] \to [t_2^*] \qquad C, \mathsf{labels}\,[t_2^*] \vdash instr_1^* : [t_1^*] \to [t_2^*] \\ (C.\mathsf{tags}[x] = [t^*] \to [])^* \\ (C, \mathsf{labels}\,(\mathsf{catch}\,[t_2^*]) \vdash instr_2^* : [t^*] \to [t_2^*])^* \\ (C, \mathsf{labels}\,(\mathsf{catch}\,[t_2^*]) \vdash instr_3^* : [] \to [t_2^*])^? \end{array}}{C \vdash \mathsf{try}\ blocktype\ instr_1^*\ (\mathsf{catch}\ x\ instr_2^*)^*\ (\mathsf{catch\_all}\ instr_3^*)^?\ \mathsf{end} : [t_1^*] \to [t_2^*]}$$

> **Note**
>
> The notation $C, \mathsf{labels}\,(\mathsf{catch}\,[t^*])$ inserts the new label type at index 0, shifting all others.

try *blocktype instr** delegate *l*

- The label $C.\mathsf{labels}[l]$ must be defined in the context.

- The block type must be valid as some function type $[t_1^*] \to [t_2^*]$.

- Let C' be the same context as C, but with the result type $[t_2^*]$ prepended to the labels vector.

- Under context C', the instruction sequence $instr^*$ must be valid with type $[t_1^*] \to [t_2^*]$.

- Then the compound instruction is valid with type $[t_1^*] \to [t_2^*]$.

$$\frac{C \vdash blocktype : [t_1^*] \to [t_2^*] \qquad C, \mathsf{labels}\,[t_2^*] \vdash instr^* : [t_1^*] \to [t_2^*] \qquad C.\mathsf{labels}[l] = [t_0^*]}{C \vdash \mathsf{try}\ blocktype\ instr^*\ \mathsf{delegate}\ l : [t_1^*] \to [t_2^*]}$$

> **Note**
>
> The label index space in the context C contains the most recent label first, so that $C.\mathsf{labels}[l]$ performs a relative lookup as expected.

rethrow *l*

- The label $C.\mathsf{labels}[l]$ must be defined in the context.

- Let $(\mathsf{catch}^?\,[t^*])$ be the label type $C.\mathsf{labels}[l]$.

- The catch must be present in the label type $C.\mathsf{labels}[l]$.

- Then the instruction is valid with type $[t_1^*] \to [t_2^*]$, for any sequences of value types t_1^* and t_2^*.

$$\frac{C.\mathsf{labels}[l] = \mathsf{catch}\,[t^*]}{C \vdash \mathsf{rethrow}\ l : [t_1^*] \to [t_2^*]}$$

> **Note**
>
> The rethrow instruction is stack-polymorphic.

4 Execution

4.1 Runtime Structure

Stack

Exception Handlers

Legacy exception handlers are installed by try instructions. Instead of branch labels, their catch clauses have instruction blocks associated with them. Furthermore, a delegate handler is associated with a label index to implicitly rethrow to:

$$
\begin{aligned}
catch \quad ::= \quad & \ldots \\
| \quad & \text{catch } tagidx \ instr^* \\
| \quad & \text{catch\_all } tagidx \ instr^* \\
| \quad & \text{delegate } labelidx
\end{aligned}
$$

Administrative Instructions

Administrative instructions are extended with the caught instruction that models exceptions caught by legacy exception handlers.

$$
\begin{aligned}
instr \quad ::= \quad & \ldots \\
| \quad & \text{caught}_n \{exnaddr\} \ instr^* \ \text{end}
\end{aligned}
$$

Block Contexts

Block contexts are extended to include caught instructions:

$$
\begin{aligned}
B^k \quad ::= \quad & \ldots \\
| \quad & \text{caught}_n \{exnaddr\} \ B^k \ \text{end}
\end{aligned}
$$

Throw Contexts

Throw contexts are also extended to include caught instructions:

$$
\begin{aligned}
T \quad ::= \quad & \ldots \\
| \quad & \text{caught}_n \{exnaddr\} \ T \ \text{end}
\end{aligned}
$$

4.2 Instructions

Control Instructions

try $blocktype \ instr_1^*$ (catch $x \ instr_2^*$)* (catch_all $instr_3^*$)$^?$ end

1. Assert: due to validation, $\text{expand}_F(blocktype)$ is defined.

2. Let $[t_1^m] \to [t_2^n]$ be the function type $\text{expand}_F(blocktype)$.

3. Let L be the label whose arity is n and whose continuation is the end of the try instruction.

4. Assert: due to validation, there are at least m values on the top of the stack.

5. Pop the values val^m from the stack.

6. Let F be the current frame.

7. For each catch clause (catch $x_i \ instr_{2i}^*$) do:

   a. Assert: due to validation, $F.\text{module.tagaddrs}[x_i]$ exists.

   b. Let a_i be the tag address $F.\text{module.tagaddrs}[x_i]$.

   c. Let $catch_i$ be the catch clause (catch $a_i \ instr_{2i}^*$).

8. If there is a catch-all clause (catch_all $instr_3^*$), then:

   a. Let $catch'^?$ be the handler (catch_all $instr_3^*$).

9. Else:

   a. Let $catch'^?$ be empty.

10. Let $catch^*$ be the concatenation of $catch_i$ and $catch'^?$.

11. Enter the block val^m $instr_1^*$ with label L and exception handler $\mathsf{handler}_n\{catch^*\}^*$.

$$
\begin{aligned}
&F; val^m \ (\mathsf{try}\ bt\ instr_1^*\ (\mathsf{catch}\ x\ instr_2^*)^*\ (\mathsf{catch\_all}\ instr_3^*)^?\ \mathsf{end} \quad \hookrightarrow \\
&\quad F; \mathsf{label}_n\{\epsilon\}\ (\mathsf{handler}_n\{(\mathsf{catch}\ a_x\ instr_2^*)^*\ (\mathsf{catch\_all}\ instr_3^*)^?\}\ val^m\ instr_1^*\ \mathsf{end})\ \mathsf{end} \\
&\quad (\text{if } \mathrm{expand}_F(bt) = [t_1^m] \to [t_2^n] \wedge (F.\mathsf{module.tagaddrs}[x] = a_x)^*)
\end{aligned}
$$

try $blocktype$ $instr^*$ delegate l

1. Assert: due to validation, $\mathrm{expand}_F(blocktype)$ is defined.

2. Let $[t_1^m] \to [t_2^n]$ be the function type $\mathrm{expand}_F(blocktype)$.

3. Let L be the label whose arity is n and whose continuation is the end of the try instruction.

4. Let H be the exception handler l, targeting the l-th surrounding block.

5. Assert: due to validation, there are at least m values on the top of the stack.

6. Pop the values val^m from the stack.

7. Enter the block val^m $instr^*$ with label L and exception handler *HANDLER_n{DELEGATE~l}*.

$$
\begin{aligned}
&F; val^m\ (\mathsf{try}\ bt\ instr^*\ \mathsf{delegate}\ l) \quad \hookrightarrow \quad F; \mathsf{label}_n\{\epsilon\}\ (\mathsf{handler}_n\{\mathsf{delegate}\ l\}\ val^m\ instr^*\ \mathsf{end})\ \mathsf{end} \\
&\quad (\text{if } \mathrm{expand}_F(bt) = [t_1^m] \to [t_2^n])
\end{aligned}
$$

throw_ref

1. Let F be the current frame.

2. Assert: due to validation, a reference is on the top of the stack.

3. Pop the reference ref from the stack.

4. If ref is ref.null ht, then:

   a. Trap.

5. Assert: due to validation, ref is an exception reference.

6. Let ref.exn ea be ref.

7. Assert: due to validation, $S.\mathsf{exns}[ea]$ exists.

8. Let exn be the exception instance $S.\mathsf{exns}[ea]$.

9. Let a be the tag address $exn.\mathsf{tag}$.

10. While the stack is not empty and the top of the stack is not an exception handler, do:

a. Pop the top element from the stack.

11. Assert: the stack is now either empty, or there is an exception handler on the top of the stack.

12. If the stack is empty, then:

a. Return the exception (ref.exn a) as a result.

13. Assert: there is an exception handler on the top of the stack.

14. Pop the exception handler $\mathsf{handler}_n\{catch^*\}$ from the stack.

15. If $catch^*$ is empty, then:

   a. Push the exception reference ref.exn ea back to the stack.

   b. Execute the instruction throw_ref again.

16. Else:

   a. Let $catch_1$ be the first catch clause in $catch^*$ and $catch'^*$ the remaining clauses.

   b. If $catch_1$ is of the form catch x l and the exception address a equals F.module.tagaddrs$[x]$, then:

      i. Push the values exn.fields to the stack.

      ii. Execute the instruction br l.

   c. Else if $catch_1$ is of the form catch_ref x l and the exception address a equals F.module.tagaddrs$[x]$, then:

      i. Push the values exn.fields to the stack.

      ii. Push the exception reference ref.exn ea to the stack.

      iii. Execute the instruction br l.

   d. Else if $catch_1$ is of the form catch_all l, then:

      i. Execute the instruction br l.

   e. Else if $catch_1$ is of the form catch_all_ref l, then:

      i. Push the exception reference ref.exn ea to the stack.

      ii. Execute the instruction br l.

   f. Else if $catch_1$ is of the form catch x $instr^*$ and the exception address a equals F.module.tagaddrs$[x]$, then:

      i. Push the caught exception caught$_n\{ea\}$ to the stack.

      ii. Push the values exn.fields to the stack.

      iii. Enter the catch block $instr^*$.

   g. Else if $catch_1$ is of the form catch_all $instr^*$, then:

      i. Push the caught exception caught$_n\{ea\}$ to the stack.

      ii. Enter the catch block $instr^*$.

   h. Else if $catch_1$ is of the form delegate l, then:

      i. Assert: due to validation, the stack contains at least l labels.

      ii. Repeat l times:

         • While the top of the stack is not a label, do:
            – Pop the top element from the stack.

      iii. Assert: due to validation, the top of the stack now is a label.

      iv. Pop the label from the stack.

      v. Push the exception reference ref.exn ea back to the stack.

      vi. Execute the instruction throw_ref again.

   i. Else:

      1. Push the modified handler handler$_n\{catch'^*\}$ back to the stack.

      2. Push the exception reference ref.exn ea back to the stack.

      3. Execute the instruction throw_ref again.

$$\cdots$$

$$\mathsf{handler}_n\{(\mathsf{catch}\ x\ instr^*)\ catch^*\}\ T[(\mathsf{ref.exn}\ a)\ \mathsf{throw\_ref}]\ \mathsf{end} \quad\hookrightarrow\quad \mathsf{caught}_n\{a\}\ exn.\mathsf{fields}\ instr^*\ \mathsf{end}$$
$$(\text{if}\ exn = S.\mathsf{exns}[a]$$
$$\wedge\ exn.\mathsf{tag} = F.\mathsf{module.tagaddrs}[x])$$
$$\mathsf{handler}_n\{(\mathsf{catch\_all}\ instr^*)\ catch^*\}\ T[(\mathsf{ref.exn}\ a)\ \mathsf{throw\_ref}]\ \mathsf{end} \quad\hookrightarrow\quad \mathsf{caught}_n\{a\}\ instr^*\ \mathsf{end}$$
$$B^l[\mathsf{handler}_n\{(\mathsf{delegate}\ l)\ catch^*\}\ T[(\mathsf{ref.exn}\ a)\ \mathsf{throw\_ref}]\ \mathsf{end}] \quad\hookrightarrow\quad (\mathsf{ref.exn}\ a)\ \mathsf{throw\_ref}$$

rethrow l

1. Assert: due to validation, the stack contains at least $l + 1$ labels.

2. Let L be the l-th label appearing on the stack, starting from the top and counting from zero.

3. Assert: due to validation, L is a catch label, i.e., a label of the form $(\mathsf{catch}\ [t^*])$, which is a label followed by a caught exception in an active catch clause.

4. Let a be the caught exception address.

5. Push the value ref.exn a onto the stack.

6. Execute the instruction throw_ref.

$$\mathsf{caught}_n\{a\}\ B^l[\mathsf{rethrow}\ l]\ \mathsf{end} \quad\hookrightarrow\quad \mathsf{caught}_n\{a\}\ B^l[(\mathsf{ref.exn}\ a)\ \mathsf{throw\_ref}]\ \mathsf{end}$$

Entering a catch block

1. Jump to the start of the instruction sequence $instr^*$.

Exiting a catch block

When the end of a catch block is reached without a jump, thrown exception, or trap, then the following steps are performed.

1. Let val^m be the values on the top of the stack.

2. Pop the values val^m from the stack.

3. Assert: due to validation, a caught exception is now on the top of the stack.

4. Pop the caught exception from the stack.

5. Push val^m back to the stack.

6. Jump to the position after the end of the administrative instruction associated with the caught exception.

$$\mathsf{caught}_n\{a\}\ val^m\ \mathsf{end} \quad\hookrightarrow\quad val^m$$

> **Note**
>
> A caught exception can only be rethrown from the scope of the administrative instruction associated with it, i.e., from the scope of the catch or catch_all block of a legacy try instruction. Upon exit from that block, the caught exception is discarded.

5 Binary Format

5.1 Instructions

Control Instructions

```
instr  ::=  ...
       |   0x06 bt:blocktype (in₁:instr)*
           (0x07 x:tagidx (in₂:instr)*)*
           (0x19 (in₃:instr)*)? 0x0B      ⇒   try bt in₁* (catch x in₂*)* (catch_all in₃*)?end
       |   0x06 bt:blocktype (in:instr)*
           0x18 l:labelidx                ⇒   try bt in* delegate l
       |   0x09 l:labelidx                ⇒   rethrow l
```

6 Text Format

6.1 Instructions

Control Instructions

The label identifier on a structured control instruction may optionally be repeated after the corresponding end, else, catch, catch_all, and delegate pseudo instructions, to indicate the matching delimiters.

$$
\begin{aligned}
\text{blockinstr}_I \quad ::= \quad & ... \\
| \quad & \text{`try' } I'{:}\text{label}_I \; bt{:}\text{blocktype } (in_1{:}\text{instr}_{I'})^* \\
& (\text{`catch' } \text{id}_1^? \; x{:}\text{tagidx}_I \; (in_2{:}\text{instr}_{I'})^*)^* \\
& (\text{`catch\_all' } \text{id}_1^? \; (in_3{:}\text{instr}_{I'})^*)^? \\
& \text{`end' } \text{id}_2^? \\
& \quad \Rightarrow \quad \text{try } bt \; in_1^* \; (\text{catch } x \; in_2^*)^* \; (\text{catch\_all } in_3^*)^? \; \text{end} \\
& \qquad (\text{if } \text{id}_1^? = \epsilon \vee \text{id}_1^? = \text{label}, \text{id}_2^? = \epsilon \vee \text{id}_2^? = \text{label}) \\
| \quad & \text{`try' } I'{:}\text{label}_I \; bt{:}\text{blocktype } (in_1{:}\text{instr}_{I'})^* \\
& \text{`delegate' } l{:}\text{labelidx}_I \; l{:}\text{labelidx}_I \\
& \quad \Rightarrow \quad \text{try } bt \; in_1^* \; \text{delegate } l \qquad (\text{if } \text{id}^? = \epsilon \vee \text{id}^? = \text{label}) \\
\text{plaininstr}_I \quad ::= \quad & ... \\
| \quad & \text{`rethrow' } l{:}\text{labelidx}_I \quad \Rightarrow \quad \text{rethrow } l
\end{aligned}
$$

7 Index of Instructions

Instruction	Binary Opcode	Type	Validation	Execution
try bt	0x06	$[t_1^*] \rightarrow [t_2^*]$	validation, validation	execution, execution
catch x	0x07		validation	execution
rethrow n	0x09	$[t_1^*] \rightarrow [t_2^*]$	validation	execution
delegate l	0x18		validation	execution
catch_all	0x19		validation	execution