8 Faults

Faults are based upon the WS-BaseFault model [WS-BF], taking on some of the lessons of [Loughran02], namely that extra information such as hostname and process is invaluable for locating which process among many has failed on a clustered system.

Faults are raised in response to errors either at the remote endpoint, in the local framework, or between the remote endpoint and other parts of the distributed system. They can be returned to callers in response to a an operation on an endpoint, or sent as part of a notification event.

All faults that will be explicitly sent are derived from WS-BaseFault faults. Service implementations may implicitly raise SOAPFault faults, as that is inherent in most implementations.

8.1 Fault Categories

8.1.1 Service Faults

These are the faults that are raised by the service. They are categorized into a hierarchy of WS-BaseFault faults. There is a base fault class `DeploymentFault`, from which all others are derived.

All Service interfaces must declare that they raise these `DeploymentFault` instances, rather than list the specific faults. This is to provide forward extensibility.

The API lists specific subclassed faults of `DeploymentFault` that may be generated by a service or received by a client. These faults represent some of the faults that a service implementation may send.

If an implementation has a fault state whose meaning matches that of the predefined fault, the predefined fault must be thrown. If this predefined fault has standard elements for embedded fault information, the implementation should fill them in. The implementation may add implementation-specific data within the `extra-data` element of the fault, to supplement this information. This extra data must not add new types to the XML namespaces of this deployment data. The XML schema and semantics of this extra data should be documented.

If an implementation creates new fault types for new fault states, these must extend the existing fault types which operations are declared as throwing. Again, these must not change the XML schemas of the deployment API; they must be in a new namespace. The new faults and XML content should be publicly documented.

8.1.2 Transport faults

Transport faults will inevitably be raised as the appropriate fault for the system. For example, the Apache Axis SOAP client raises `AxisFault` faults for all SOAP events, wrapping stack trace and even HTTP Fault data within the fault as DOM elements. .NET WSE has a similar fault class.

8.1.3 Relayed Faults

Relayed faults are those received by the far end and passed on. They may be WS-BaseFault Faults; HTTP error codes, SOAP faults, native language faults wrapped as SOAPFaults, or predefined deployment faults.

WS-BaseFault uses fault nesting for relaying faults; however, all faults must be a derivative of WS-BaseFault. This is addressed by defining a new WS-BaseFault derivative, a `WrappedSOAPFault`. This type is actually an extension of `DeploymentFault`. This fault can nest any received SOAPFault, with an element containing the received XML data. Well-known elements in this fault data (such as the Apache Axis stack trace and HTTP fault code) should be copied into any fields in the main fault which fill the same role.

8.2 Fault Security

Sites offering deployment services, may, for security reasons, wish to strip out some information, such as stack trace data. Implementations should provide a means to enable such an action prior to transmitting faults to callers.

Hostname and process information may be viewed as sensitive, yet again, this is exceedingly useful to operations. Implementations may provide a means to disguise this information, so that it does not describe the real hostname or process ID of a process, but instead pseudonyms that can still be used in communications with any operations team.

8.3 Internationalization

The WS-BaseFault specification makes no statement upon which language error descriptions are in.

If an implementation can return descriptions in one language, it must use xml:lang attributes to indicate the language of a description. Multiple descriptions, in different languages may be included. The client application should extract the description(s) whose language is the nearest match to that of the client.

8.4 Extending Faults

If an implementation adds new operations or properties at the existing endpoints, these new operations may raise whatever faults they see fit, within the constraints of the WS-BaseFaults specification. The implementation must not add new types to the deployment API namespace. If possible, faults raised from these new endpoints should extend the `api:DeploymentFault` type, or extended types if appropriate.

Implementations may extend the faults defined in the deployment API, following the procedure defined in the WS-BF specification. Such extended types must not be declared within the namespace of the deployment API. The extended information must be informative, to the extent that an caller that only knows the base types must still be able take appropriate action on the failure.

Client applications must be written expecting such extensions.

8.5 Fault Type Declarations

The fault hierarchy is shown in Figure 3.

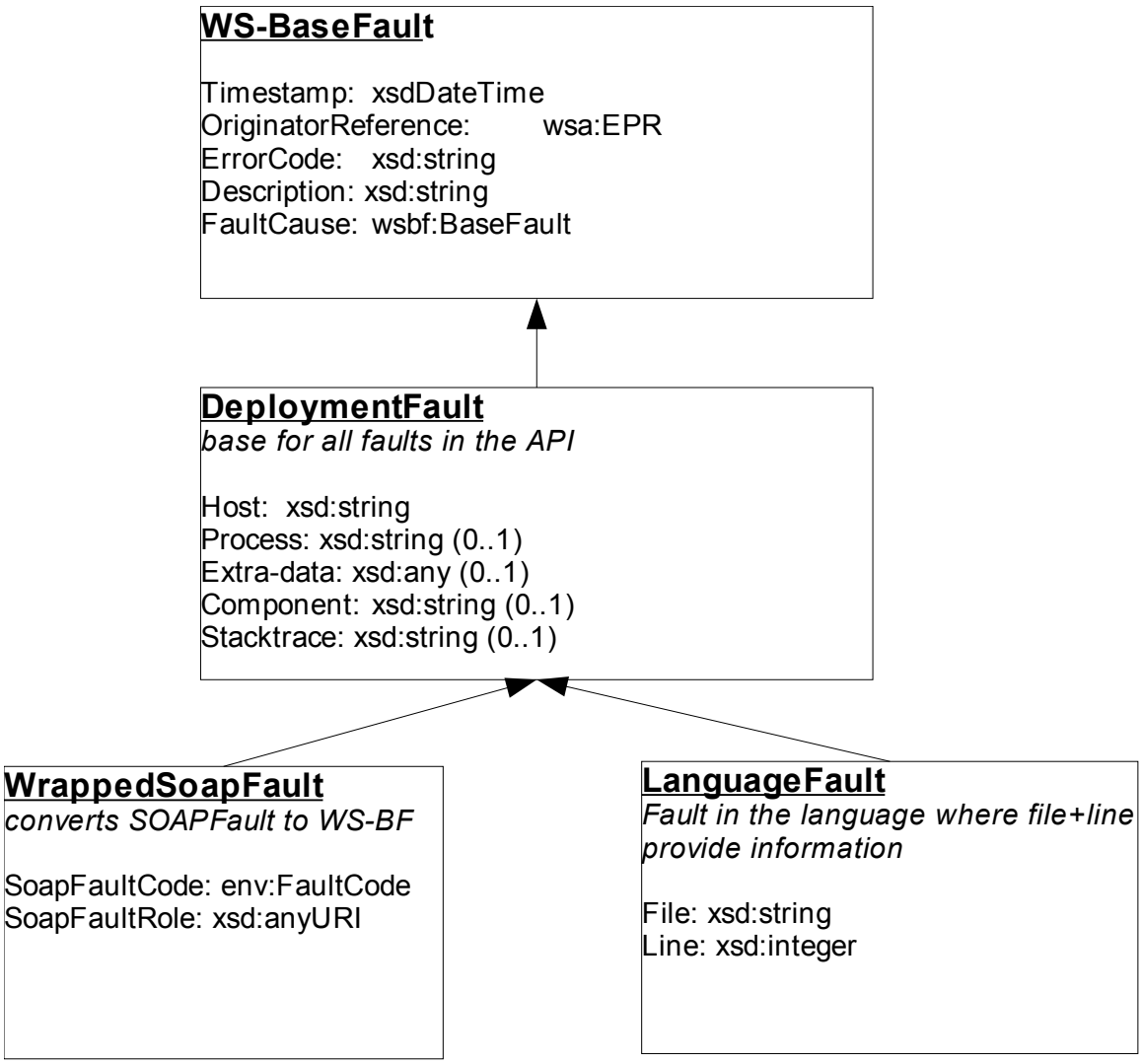

Figure 3 Fault Hierarchy

8.5.1 DeploymentFault

This type represents any fault thrown during deployment. All endpoint operations must declare that they throw this fault, and must not declare that they throw any derivative fault.

Element	*Type*	*Meaning*
`Host`	`xsd:string`	Hostname or pseudonym
`Process`	`xsd:string`	Any process identifier suitable for diagnostics
`Extra-data`	`xsd:any`	Extra fault data

Element	*Type*	*Meaning*
`Component`	`xsd:string`	Path to component raising the fault
`StackTrace`	`xsd:string`	Stack trace of fault

Implementations must include a component reference if it is known. Implementations should include hostname and process information.

8.5.2 LanguageFault

A language fault represents any fault in language processing for which a file and line number are relevant.

Element	*Type*	*Meaning*
`File`	`xsd:string`	Filename/URI of file at fault
`Line`	`xsd:integer`	Line number within the file

This information must be included if it is known.

If a deployment request includes the deployment descriptor inline, the file and line info is missing/wrong. What to do?

8.5.3 WrappedSOAPFault

This type represents a mapping of a classic W3C-style SOAPFault [SOAP1.2] to a WS-BaseFault, as an extension of `DeploymentFault`. It adds two new elements to contain data unique to SOAPFaults.

Element	*Type*	*Meaning*
`SoapFaultCode`	`env:FaultCode`	Fault code information
`SoapFaultRole`	`xsd:anyURI`	Role of sender

The normative mapping of SOAPFault elements to `WrappedSOAPFault` elements is as follows:

SOAP1.2	*WrappedSOAPFault*
`/env:Code`	`/api:SoapFaultCode`
`/env:Role`	`/api:SoapFaultRole`
`/env:Detail`	`/api:ExtraData`
`/env:Reason/env:text`	`/wsbf:Description`

Any text elements under `env:Reason` must be converted into separate description elements in the fault; all `xml:lang` attribute must be preserved.

Detail from SOAP stacks with well-known fault fields, such as the Apache Axis stack trace, may be imported into appropriate fields in the `DeploymentFault`.

An example mapping for this is shown in the following table, a table in which the namespace `axis` is `http://xml.apache.org/axis/` :

Axis1.2	*WrappedSOAPFault*
`/env:Detail/axis:stackTrace`	`/api:StackTrace`
`/env:Detail/axis:hostname`	`/api:Host`
`/env:Detail/axis:HttpErrorCode`	`(unmapped)`

The Axis bindings are purely an implementation specific feature, and currently documented only in the Axis source (in `AxisFault.java` and `Constants.java`). However, as Apache Axis is a common SOAP implementation in the Java community, implementations should extract the data if it is present, and map it into the appropriate fields.

8.6 Fault Error Codes

Specific fault error codes, and their meaning, are covered in a separate document.

Every unique fault will be described by its own fault code. Deployment faults that are part of the API specification will all be in the namespace http://X/Y/Z with their code value described in the CDDLM Fault Specification.

TODO: Fault Specification, namespace

Implementations may add new fault codes in different namespaces. They must not add new fault codes to the primary fault namespace of the deployment API.