z/TPF EE V1.1
z/TPFDF V1.1
TPF Toolkit for WebSphere® Studio V3
TPF Operations Server V1.2

IBM Software Group

TPF Users Group Fall 2006

TPF XML API

Name: Barry M. Baker
Venue: Distributed Systems Subcommittee

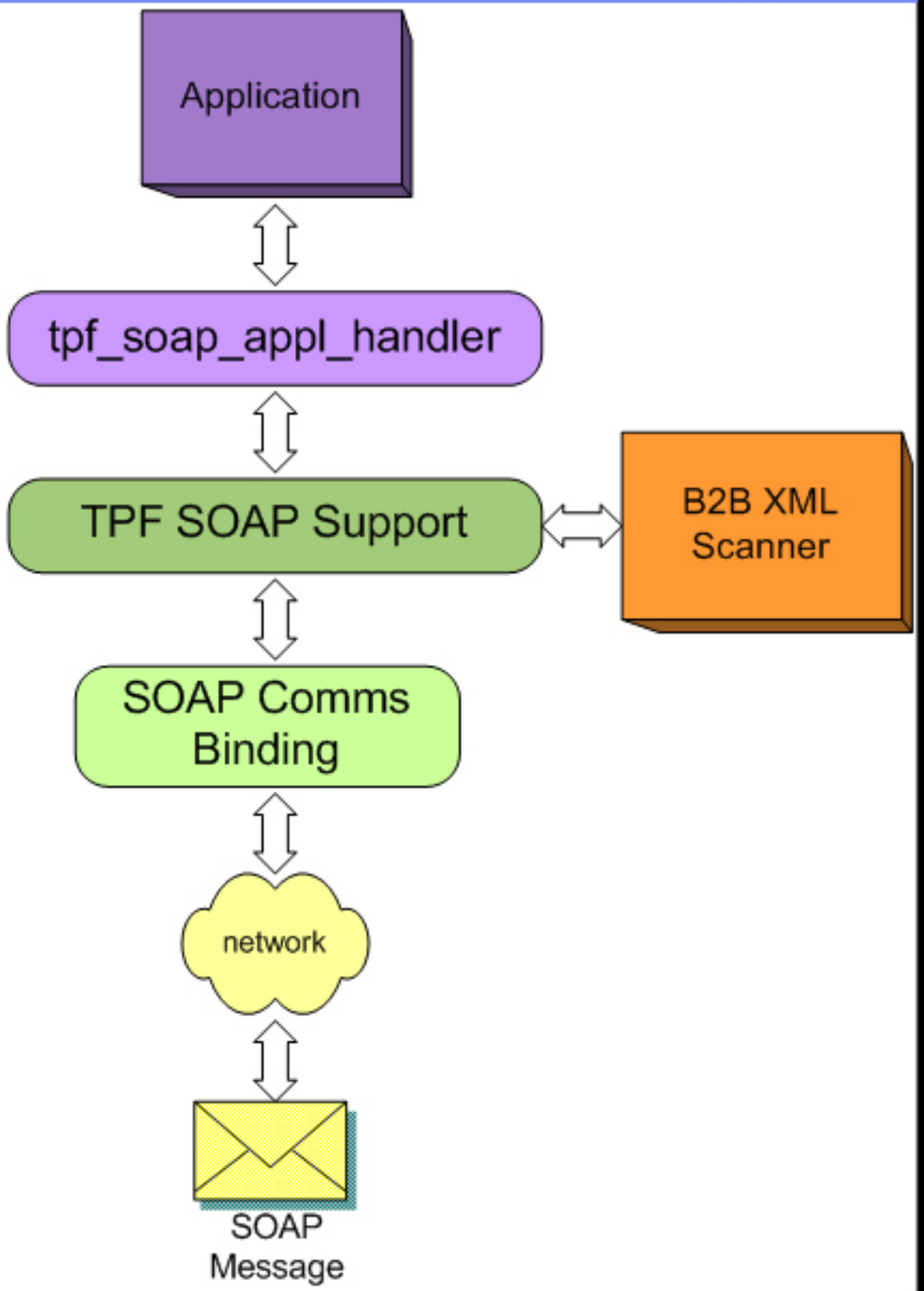
AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Agenda

- History and Context
- TPF XML API:
  - What is it?
  - Logical Structure and Terminology
- When would you want to use the TPF XML API?
  - Processing SOAP messages
  - Creating an XML document
  - Parsing an XML document and accessing the data
  - Modifying a parsed XML structure and pass it to another application for further processing
- Restrictions
- TPF XML API versus XML4C

TPF XML API: History and Context

- TPF SOAP Support
  - tpf_soap_handler(): an external function to pass a SOAP message to the TPF SOAP Support. SOAP Communications Bindings can be written for any transport to make use of this function to pass a message to the TPF SOAP Support
  - tpf_soap_appl_handler: a user exit allowing the user to specify how to route a SOAP message to the appropriate application
  - This support requires the user to have both a detailed understanding of XML and of the TPF specific data structure representing parsed XML.
- At the Spring 2005 TPFUG we presented a list of the APIs we were considering for implementation
  - At first we were going to provide this function in the context of SOAP, but later decided to open up access to the B2B XML scanner and these APIs to all Applications
- We had our High Level Design reviewed by a number of customers

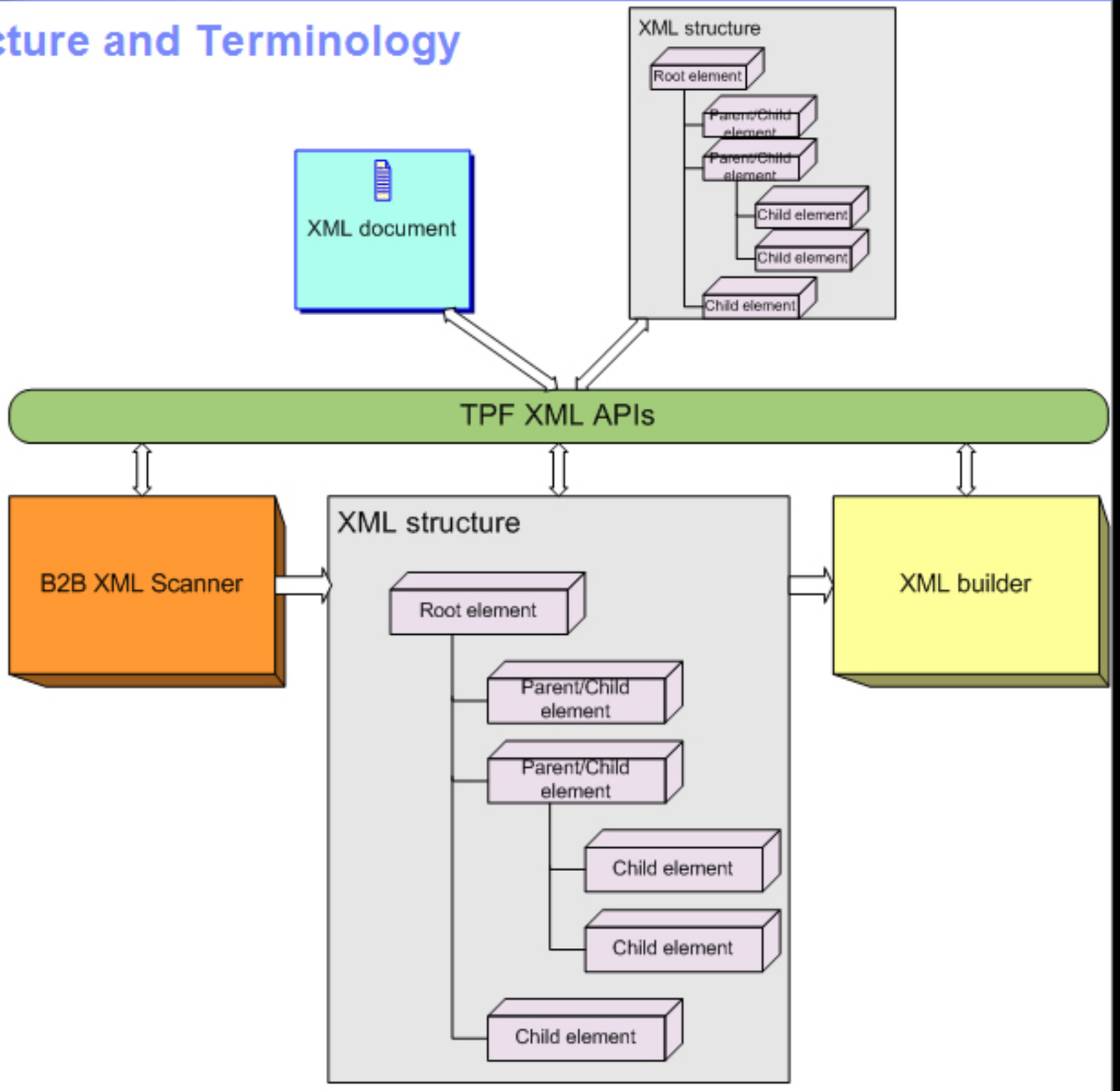


TPF XML API Support

- z/TPF and TPF4.1
  - PJ30866 - z/TPF PUT 2
  - PJ30936 - TPF4.1 PUT20
- What is the TPF XML API?
  - A set of XML APIs that applications can use to process and create XML documents without knowledge of the underlying data structure (infoNodes) produced by the B2B XML Scanner. The TPF XML APIs are conceptually based on the Document Object Model (DOM).
  - An abstraction layer: Future XML parser support can be added under the TPF XML API, minimizing the application migration effort to a new parser technology
- Benefits of Support
  - Less Coding
    - No longer need to write code to traverse the infoNodes structure, using string functions to locate the data
  - Data Conversion
    - Provide for the conversion from XML Schema Datatypes to C/C++ data types
  - Ability to create XML documents

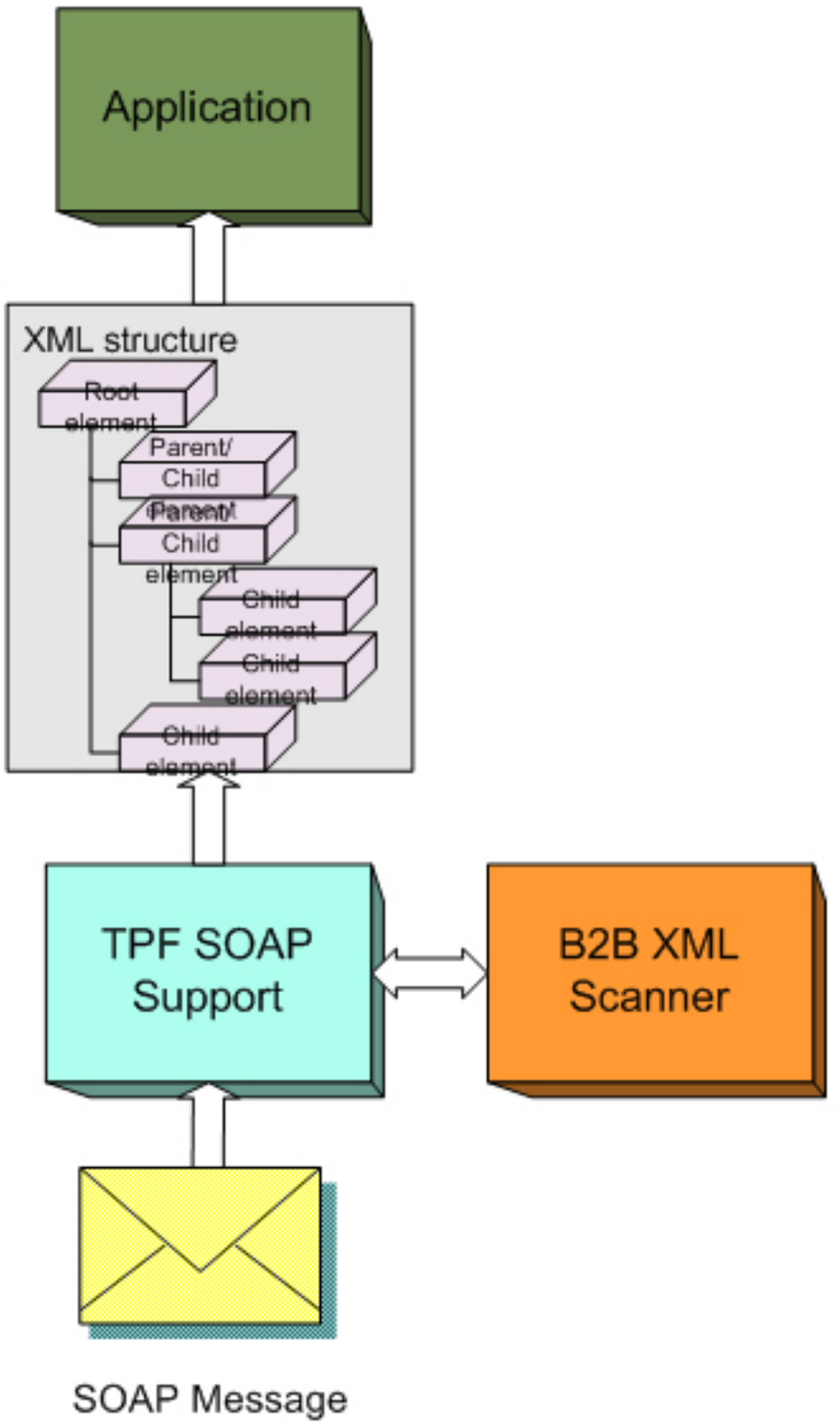
TPF XML API: Logical Structure and Terminology

- **XML document:** area of main storage containing, in text, the self-describing data using XML markup
- **XML structure:** output of an XML Scanner/Parser, represents XML documents in a tree-like structure
- **TPF XML APIs:** subset of DOM-like APIs that provide for processing/creating XML documents/structures
- **B2B XML Scanner:** Provides for the parsing of XML documents into XML structures and is the only scanner/parser supported by the TPF XML APIs at this time
- **XML builder:** Provides for the creation of XML documents from XML structures



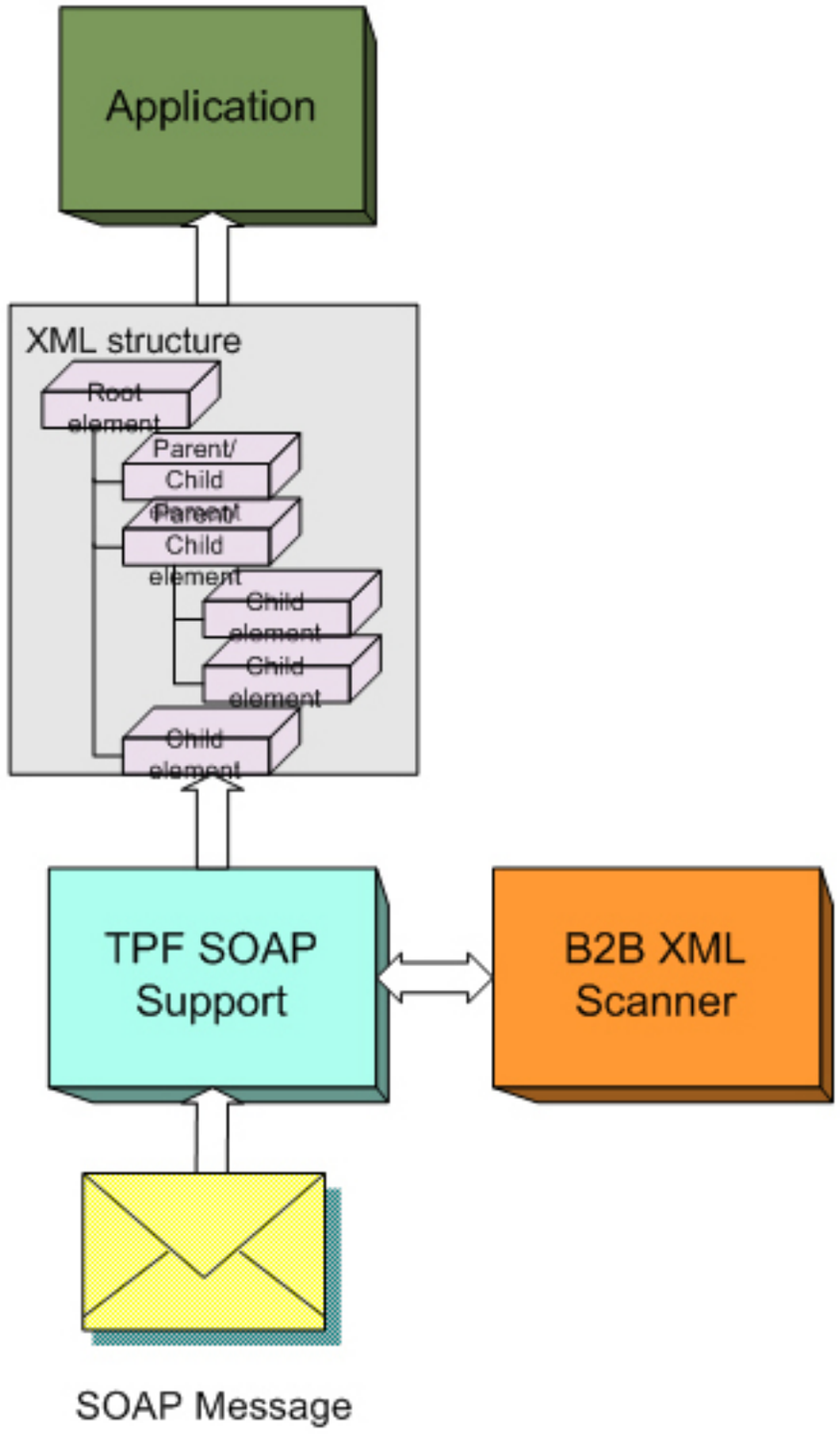
TPF XML API: Processing SOAP Messages

- Interface between TPF SOAP support and an Application includes an XML structure that is created by the B2B XML Scanner and the application is responsible for working with the lower-level structure (infoNodes) and must contain code to **traverse** this structure and **access/convert** the data.
- This interface is unchanged by this support but allows your applications to use the TPF XML APIs to operate on the SOAP input message at a higher level of abstraction as an alternative to coding directly to the infoNodes data structure.
- TPF XML APIs of interest:
  - tpf_xml_initialize_handle()
  - tpf_xml_getFirstElementByTagName()
  - tpf_xml_getNextElement()
  - tpf_xml_XSD_conversion()
- The application is responsible for creating the complete SOAP response message. The TPF XML APIs also help with this task, relieving the application developer from having to understand many of the details about XML mark-up (discussed further on the next slide)



TPF XML API: Processing SOAP Messages

- Interface between TPF SOAP support and an Application includes an XML structure that is created by the B2B XML Scanner and the application is responsible for working with the lower-level structure (infoNodes) and must contain code to **traverse** this structure and **access/convert** the data.
- This interface is unchanged by this support but allows your applications to use the TPF XML APIs to operate on the SOAP input message at a higher level of abstraction as an alternative to coding directly to the infoNodes data structure.
- TPF XML APIs of interest:
  - tpf_xml_initialize_handle()
  - tpf_xml_getFirstElementByTagName()
  - tpf_xml_getNextElement()
  - tpf_xml_XSD_conversion()
- The application is responsible for creating the complete SOAP response message. The TPF XML APIs also help with this task, relieving the application developer from having to understand many of the details about XML mark-up (discussed further on the next slide)



Comparing Current Support and the TPF XML API Support in the Context of SOAP

- Current Support
  - Finding an Element

```
dptr = ((infoNodes *) *(info))->docnode_ptr;
eptr = ((infoNodes *) *(info))->element_ptr;
aptr = ((infoNodes *) *(info))->attribute_ptr;
cptr = ((infoNodes *) *(info))->content_ptr;
nptr = ((infoNodes *) *(info))->namespace_ptr;

for (i = 0; i < dptr->numElements; i++, ++eptr)
{  if(strncmp("number1",eptr->localName,
          sizeof("number1")) == 0);
  {  foundElement = 1;
     break;
  }
}
/* not done yet,now you need to dereference into
content array to get at the value as a string */
```

  - Adding an Element to an XML structure

No Support for creating/updating either XML structures or XML documents

- TPF XML API
  - Finding an Element **and converting the data**

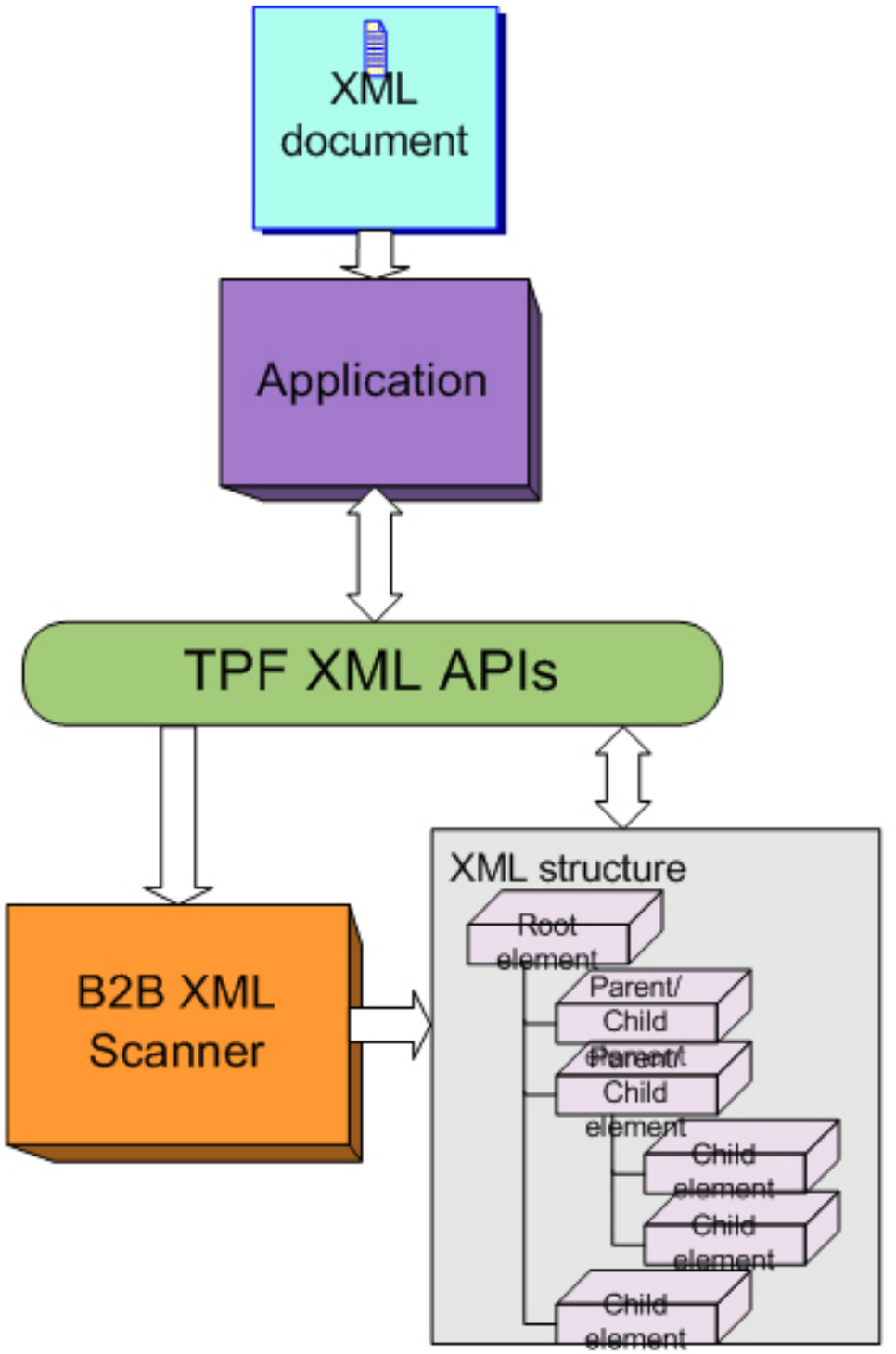
```
retCode = tpf_xml_initialize_handle(&inputHandle,
                B2B_XML_SCANNER,(void *)xml_ptr);

xptr =
tpf_xml_getFirstElementByTagName(inputHandle,
                                 TYPE_INT,"number1");
```

  - Adding an Element to an XML structure

```
retCode = tpf_xml_appendElement(outputHandle,
            "Body","Calculator",NULL,"ns1",
            "http://www.ibm.com/tpf/CalcService",
            1,"result",resultstring, NULL,NULL);
```

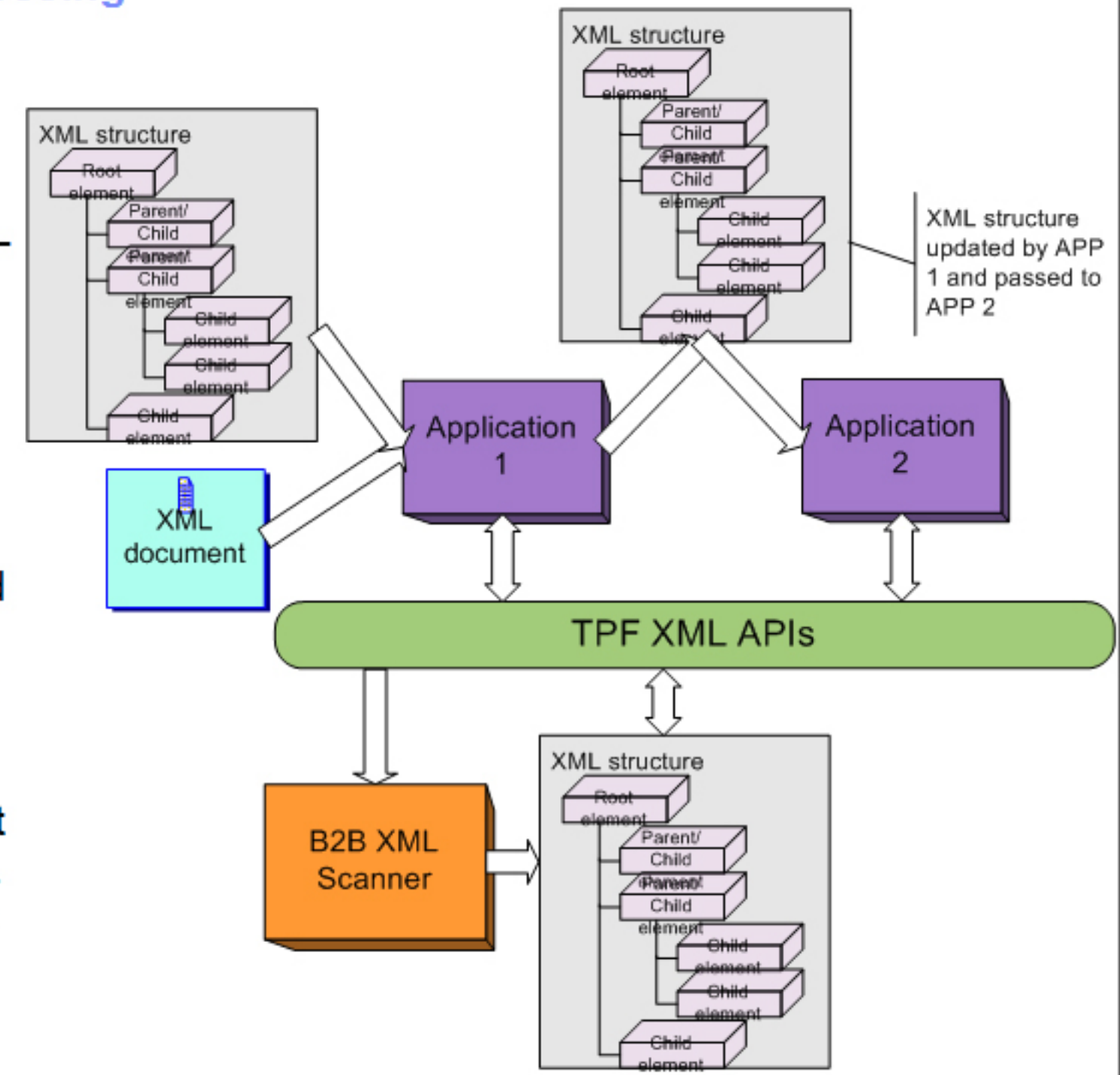
TPF XML API: Parsing an XML document and accessing the data

- Outside the scope of the TPF SOAP support, the TPF XML APIs can be used to process any XML document. The application will need to have the entire XML document in main storage and then it can use the TPF XML APIs to parse the document, to create the XML structure, and then traverse the structure to access the elements of interest and optionally convert the data based on the known data type.
- TPF XML APIs of interest:
  - tpf_xml_initialize_handle()
  - tpf_xml_parseDocument()
  - tpf_xml_getFirstElementByTagName()
  - tpf_xml_getNextElement()
  - tpf_xml_XSD_conversion()



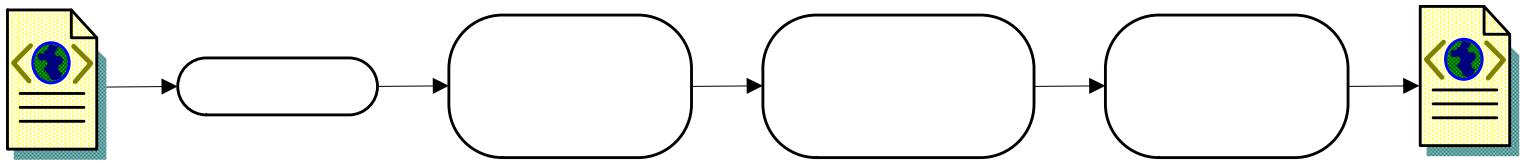
TPF XML API: To Modify a parsed XML document structure/data and pass it to another application for further processing

- TPF XML APIs can be used in many other ways
- One shown to the right includes application 1 either starting from an XML structure or an XML document and making updates to the XML structure and passing that along for further processing by another application
- Alternatively (not shown here), application 1 could have created a portion of the XML structure and passed the structure along to application 2, where the creation of the XML structure could be completed. Once complete, the XML builder can be called to traverse the XML structure and write out to a buffer the complete XML document. This usage may be useful in the context of SOAP message processing.

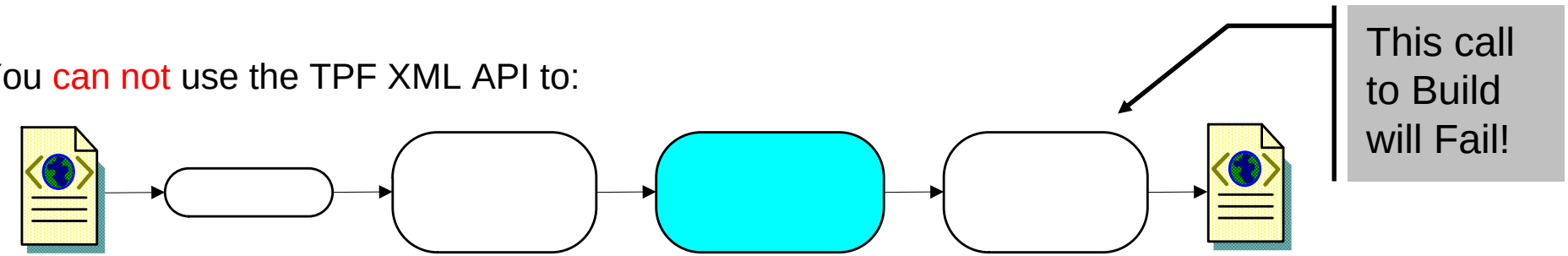


Restrictions

- XML documents that are parsed into XML structures using the TPF XML APIs can not then be used by the XML builder to create XML documents due to *normalization* (see www.w3.org) processing done by the B2B XML Scanner (note: XML4C has a similar restriction)
  - You can use the TPF XML API to (See backup slides for a SOAP example of this):



  - You can not use the TPF XML API to:



- Because the XML structure is maintained in ECB heap, it can not be shared across ECBs

TPF XML API and XML4C

- Analogy:
  - A Sports Car versus a Sport Utility Vehicle
  - The TPF XML API provides a thin layer of abstraction on top of the data structure used by the B2B XML Scanner, which is used in the TPF SOAP support for its performance benefits (processing time and memory usage) over XML4C.

Functional Comparison	well-formedness checking	DTD validation	XML Schema validation	Namespaces	SAX 1.0	SAX 2.0	DOM Level 1	DOM Level 2
TPF XML API	Yes	No	No	Yes	No	No	subset (conceptual)	subset (conceptual)
XML4C	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

- Performance
  - Limited performance testing
    - Comparing the parsing costs (DOM/non-validating), which can be a significant aspect of processing an XML document, we found the B2B XML Scanner using 10-50% of the processing time of XML4C.
      - **NOTE**: These numbers are not a guarantee of the results that you will see. Your actual performance will depend on a combination of your XML documents, application and configuration.

Questions?

Backup

Complete SOAP Example

- Simple Calculator Example:
  - Input SOAP Message : two operands and an operation

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2002/06/soap-envelope">
  <SOAP-ENV:Body>
     <ns1:calculator xmlns:ns1="http://www.ibm.com/tpf/CalcService">
            <ns1:operation>add</ns1:operation>
            <ns1:number1>10</ns1:number1>
            <ns1:number2>20</ns1:number2>
     </ns1:calculator>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

  - Output SOAP Message: result

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--Sample application - TPF XML APIs-->
<!--Calculator-->
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2002/06/soap-envelope">
  <SOAP-ENV:Body>
     <ns1:calculator xmlns:ns1 ="http://www.ibm.com/tpf/CalcService">
            <ns1:result>30</ns1:result>
     </ns1:calculator>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Complete SOAP Example – Handling the Request

```
XMLHandle     inputHandle = 0, outputHandle = 0;
infoNodes     *xml_ptr = *info;
int           retCode = 0, number1 = 0, number2 = 0, result = 0, len = 0;
char          resultstring[32];
char          *operation = NULL, *output = NULL;
xmlNodesArray *xptr = NULL;

/* initialize handle for use with the XML APIs                */
/* this handle is used for all APIs operating on the request */
retCode = tpf_xml_initialize_handle(&inputHandle,B2B_XML_SCANNER,(void *)xml_ptr);

/* use getFirstElement to find the operation */
xptr = tpf_xml_getFirstElementByTagName(inputHandle,TYPE_TEXT,"operation");

/* get the first operation */
if (xptr == NULL || xptr->returnCode == 0)
{
   create_soap_fault_return(SenderOrClient,
                            "Unable to perform calculator function",
                            "Expected element tag not found");
   return SendErrorReplySender;
}else {
 operation = &(xptr->nodesArray[0].nodeValueStr);
}

...
```

Input to this function:

-infoNodes **info

-soapMsg *inMsg

-soapMsg *outMsg

Set up a handle

Get parameter data

Note the data type parameter

The data is always available as a string regardless if conversion was performed.

Complete SOAP Example – Handling the Request (cont)

```
/* use getFirstElement to find 1st number to be used by calculator */
xptr = tpf_xml_getFirstElementByTagName(inputHandle,TYPE_INT,"number1");

/* get the first operand */
if (xptr == NULL || xptr->returnCode == 0)
{
   create_soap_fault_return(SenderOrClient,
                            "Unable to perform calculator function",
                            "Expected element tag not found");
   return SendErrorReplySender;
}else {
   number1 = xptr->nodesArray[0].nodeValueDataType.nodeValueInt;
}

/* use getNextElement to find 2nd number to be used by calculator */
xptr = tpf_xml_getNextElementByTagName(inputHandle,TYPE_INT, "number2");

/* get the second operand */
if (xptr == NULL || xptr->returnCode == 0)
{
   create_soap_fault_return(SenderOrClient,
                            "Unable to perform calculator function",
                            "Expected element tag not found");
   return SendErrorReplySender;
}else {
   number2 = xptr->nodesArray[0].nodeValueDataType.nodeValueInt;
}

result = calculator(number1, number2, operation);
sprintf(resultstring,"%d",result);
...
```

Get parameter data

Note the data type parameter

Get parameter data

Pass the data to the application

Complete SOAP Example – Building the Response

```
/* we're done with the handle used for processing the request */
retCode = tpf_xml_terminate_handle(&inputHandle);

/* this handle is used for all APIs to create/build the response */
retCode = tpf_xml_initialize_handle(&outputHandle,NO_PARSER,NULL);

/* createXMLstructure creates a new XML structure for the response */
retCode = tpf_xml_createXMLstructure(outputHandle,"1.0",819,"ISO-8859-1",
                                     STANDALONE_YES,2,
                                     "Sample application - TPF XML APIs",
                                     "Calculator");

/* add the first required element of the SOAP response - Envelope */
retCode = tpf_xml_appendElement(outputHandle,NULL,"Envelope",NULL,"SOAP-ENV",
                                "http://www.w3.org/2002/06/soap-envelope",0);

/* add the next required element of the SOAP response - Body */
retCode = tpf_xml_appendElement(outputHandle,"Envelope","Body",NULL,"SOAP-ENV",
                                "http://www.w3.org/2002/06/soap-envelope",0);

/* add the element for the result of the calculator function */
retCode = tpf_xml_appendElement(outputHandle,"Body","Calculator",NULL,"ns1",
                                "http://www.ibm.com/tpf/CalcService",
                                1, "result", resultstring, NULL, NULL);

/* call builder to create an XML document from the XML structure */
output = (char *)calloc(1, 3000);
output = tpf_xml_buildXMLdocument(outputHandle, &len, 0);
...
```

Free the handle

Set up a handle to create an XML structure

Create the XML structure

Add the elements to the structure

Build the XML document from the XML structure

Complete SOAP Example – Building the Response (cont)

```
/* set up the response message structure passed by TPF SOAP */
outMsg->XMLptr = output;
outMsg->msgLength = len;

/* we're done with the handle used for processing the response */
retCode = tpf_xml_terminate_handle(&outputHandle);

/* return to TPF SOAP indicating that a response message has been */
/* built and is attached to the outMsg structure                  */
return(SendReply);

}
```

Set up the TPF SOAP response structure

Do not need the handle any longer

Return to TPF SOAP

Trademarks

IBM is a trademark of the International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries

Other company, product, or service names may be trademarks or service marks of others.

Notes

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.