z/TPF V1.1

SOA Update
New Functionality in z/TPF PUT 7

Lisa Banks
SOA Subcommittee

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

Agenda

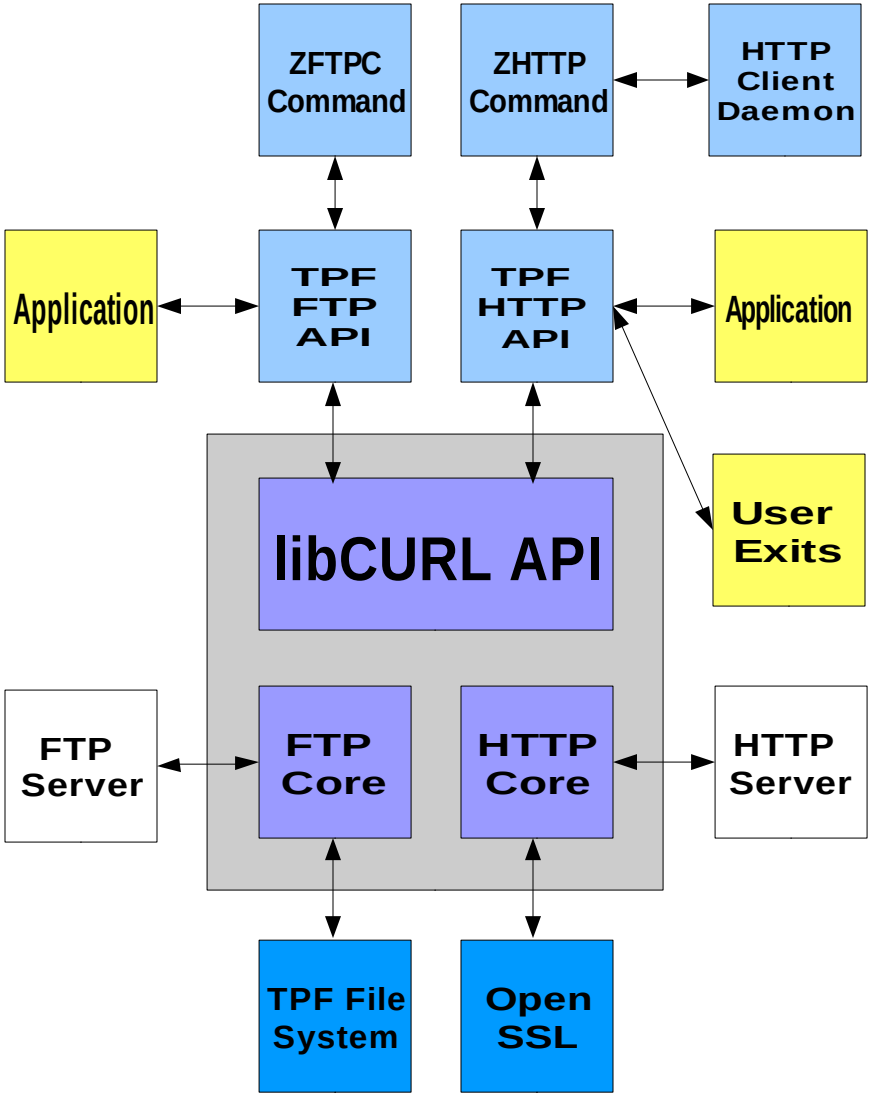
- **HTTP Client Enhancement**
  - libCURL Update
  - Sub-second Timeout for SOAP Consumer
- SOAP Message Handler Enhancement
- XML Encryption Support

New Version of libCURL

- **The previous port of libCURL was version 7.16.1**
- **A new port has been done to support the 7.19.3 version of libCURL**

libcurl – Open Source URL Transfer Library

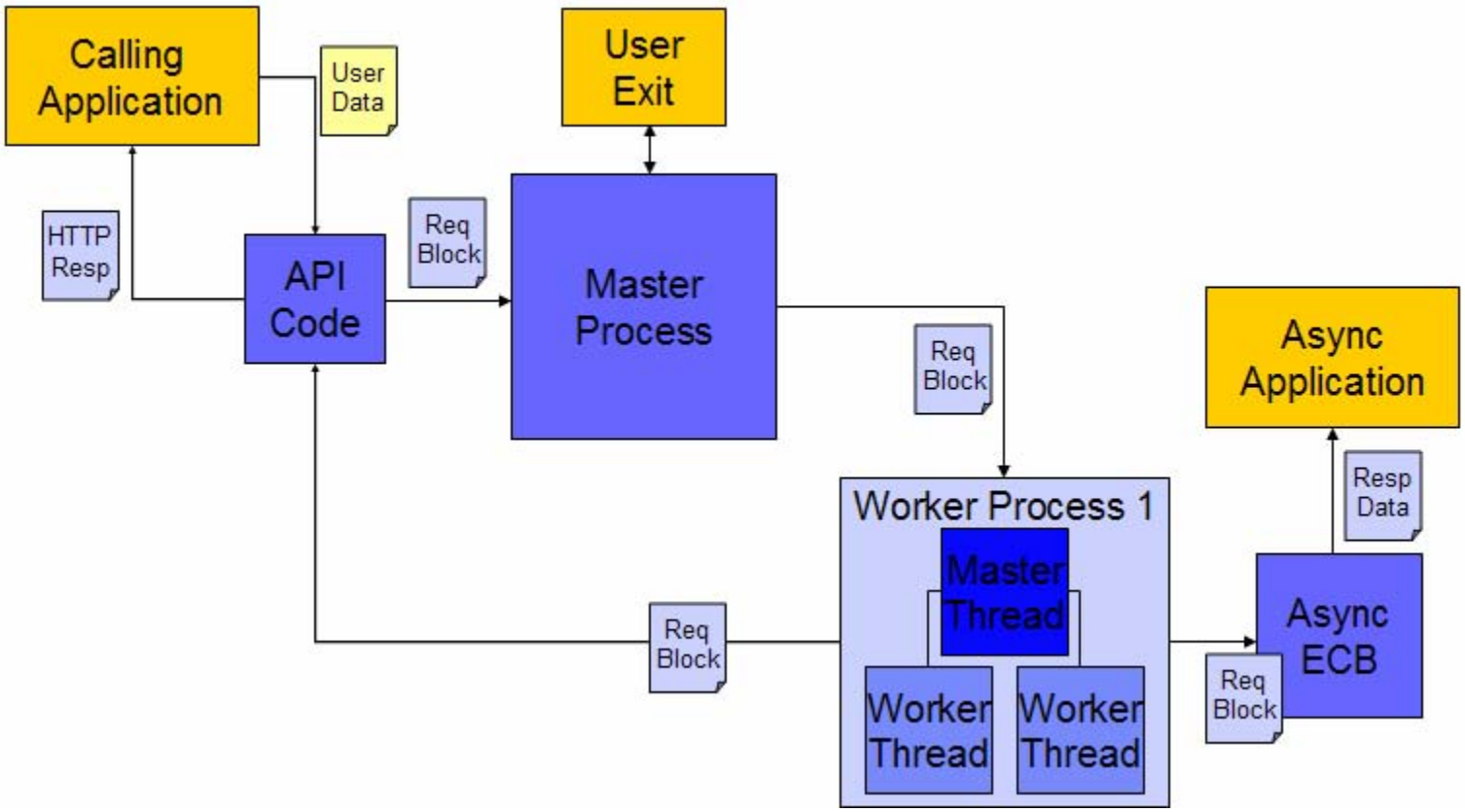
- **z/TPF's FTP and HTTP client support uses libcurl to handle the protocols and the actual transfer of files from one system to another**
- **libcurl is an open source library that provides client APIs for transferring files**
  - Supports FTP and FTPS (using OpenSSL)
  - Supports HTTP and HTTPS (using OpenSSL)
  - For additional information about libcurl, see http FTP and FTPS (using ://curl.haxx.se/libcurl
- **The libcurl library is not shipped with the z/TPF product**
  - If you wish to use the FTP or HTTP client, you must download libcurl following the instructions from the z/TPF Downloads website
  - If libcurl is *not* installed, then both the FTP and HTTP client APIs and the commands are disabled



How Are These Enhancements Be Incorporated Into z/TPF?

- **HTTP enhancements are built on z/TPF's HTTP Client Support**
  - APAR PJ32013 (PUT 4)
    - libCURL library -- APAR PJ32052
  - HTTP Client APIs
    - tpf_httpInitHandle
      - This function initializes a HTTP client handle to be used for subsequent HTTP client requests.
    - tpf_httpPerform
      - This function sends an HTTP client request to a remote HTTP server, using a previously established HTTP client handle.
    - tpf_httpEnd
      - This function ends an HTTP) client handle with a remote HTTP server.
    - tpf_httpPerform1
      - This function performs a single HTTP client request
      - The tpf_httpPerform1 function establishes an HTTP connection, performs the request, and ends the connection.
    - tpf_httpGetOpts–Get options for an HTTP client connection
      - This function returns the options that were set for a previously established HTTP client handle.
      - It returns connection information that was set by the tpf_httpInitHandle function and request information for the last tpf_httpPerform request that was made.

Application Flow of an HTTP Client Request



How Fast Can I Timeout?

- **HTTP Client and SOAP Consumer Support can now handle millisecond granularity**
  - Previous support only allowed second granularity

What Is Asynchronous HTTP Support?

- **Allow applications to request that an HTTP response message be processed by a new ECB.**
- **Once an HTTP request is scheduled to be sent, control is returned immediately to the calling application**

Why Do I Need Asynchronous HTTP Support?

- **Applications can leverage HTTP client support without blocking for the request to be sent or for the response to be received**
- **This allows applications to proceed with other processing**
- **Applications can also exit their ECBs to free system resources**

How Do I Make An Asynchronous HTTP Request?

- **A new version of the t_httpRequest structure is passed as input to the following APIs**
  - tpf_httpPerform()
  - tpf_httpPerform1()
- **The new version of t_httpRequest contains additional fields to specify asynchronous options**

How Do I Make An Asynchronous HTTP Request?

- **requestVersion**
  - Indicates the version of the structure
    - HTTPC_VERSION_1
      - Current version
    - **HTTPC_VERSION_2**
      - **New version with extended option fields**
- responseType
  - **The type of response processing to be performed**
    - **HTTPC_SYNC**
      - **Synchronous response requested**
    - **HTTPC_ASYNC**
      - **Asynchronous response requested**

How Do I Make An Asynchronous HTTP Request?

- **asyncProgram**
  - The name of the z/TPF real-time program to activate after the HTTP response has been received
- **asyncData**
  - Used to pass a variable amount of data to the new ECB
- **asyncDataLength**
  - The length of the data referenced by the asyncData field
- **asyncIS**
  - The I-stream number where the new ECB should be created

How Do I Process An Asynchronous Response?

- **Processing is implemented in C shared objects**
- **CSO must have a single entry point**
- **The interface to the CSO must be as follows:**
  - void ABCD (t_httpAsyncParms *asyncParms);

What Information Will The Asynchronous Program Receive?

- **httpOptions**
  - Information about the HTTP client connection handle that was used to make the request
- **httpReturnCode**
  - The return code from the HTTP client support
- **httpResponse**
  - The ECB heap address of the response message from the HTTP server
- **data**
  - A copy of the request data
- **dataLength**
  - The length of the data referenced by the data field
- **headerList**
  - A copy of the HTTP header list sent with the request
- **headerNum**
  - The number of HTTP headers referenced by the headerList field
- **asyncData**
  - A copy of the user data passed from the requesting application
- **asyncDataLength**
  - The length of the data referenced by the asyncData field

What Happens When I Make An Asynchronous HTTP Request?

1. **The request is sent to the HTTP client daemon which submits the HTTP request on behalf of the application**
2. **Control is return to the application once the daemon has successfully scheduled the HTTP request**
3. **The daemon will “wait” for a response from the HTTP session**
4. **Once the response is received or the request times out, the daemon will created a new ECB and enter the program the application specified to gain control**

What Is Shared HTTP Session Support?

- **Ability to create an HTTP session with a persistent connection to a remote destination.**
- **This HTTP session is tied to an HTTP client daemon ECB (process) rather than an application's ECB (process)**
- **This allows the HTTP session to be used by multiple ECBs (processes).**

Why Do I Need Shared HTTP Sessions?

- **Previously, an HTTP session was tied to a single ECB (process). If HTTP requests to the same destination were made by multiple ECBs (processes) each ECB (process) would have to establish an HTTP connection to that destination.**
- **This creates overhead by repeatedly starting and ending TCP/IP sessions for a single request/reply. The use of HTTPS dramatically increases this overhead due to SSL handshakes.**

How Do I Establish Shared Sessions

- **Shared Session file**
  - XML file that resides in the file system
  - The format is defined in the by a provided XML schema
  - Lists the connection information for each session

```
<ztpf:session sessionsToStart="1" maxSessions="12" maxQueueLen="21">
   <ztpf:connectTimeout>1221</ztpf:connectTimeout>
   <ztpf:URL>http//www.example.com</ztpf:URL>
   <ztpf:netrc>/.netrc</ztpf:netrc>
   <ztpf:proxy name="http://xx.xx.xx.xxx">
       <ztpf:proxyPort>12</ztpf:proxyPort>
       <ztpf:proxyUserPassword>llll:bbbb</ztpf:proxyPort>
   </ztpf:proxy>
   <ztpf:allowRedirects>false</ztpf:allowRedirects>
 </ztpf:session>
```

Shared Session File Elements

- session
  - Identifies each HTTP client session. The following elements are children of this element: connectTimeout, netrc, proxy, URL, and allowRedirects. The following attributes are required for this element: sessionsToStart, maxSessions, and maxQueueLen.
  - **sessionsToStart**
    - This attribute must be present on the session element of the HTTP shared session XML deployment. It indicates the number of connections to be started initially. The maximum value is 99.
  - **maxSessions**
    - This attribute must be present on the session element of the HTTP shared session XML deployment. It indicates the maximum number of connections for this session. The maximum value is 99.
  - **maxQueueLen**
    - This attribute must be present on the session element of the HTTP shared session XML configuration file. It indicates the maximum number of HTTP requests to that can be queued when all shared session connections to the destination are in use. When a shared session connection becomes available, an HTTP request is pulled from the top of the queue. If the maxQueueLen is exceeded, additional requests are discarded and the ETPFHTTP_QUEUEMAX errno is returned. The maximum value is 99.

Shared Session File Elements

- connectTimeout
  - An integer that indicates the amount of time in milliseconds to wait for a connect request to complete. This is an integer in the range of 1 to 65535.
- netrc
  - Indicates the fully-qualified path name of the .netrc file to be used when determining the user name or password, or both. When this field is set to NULL, the .netrc file is not checked. The maximum length of this string is 1024 bytes.
- URL
  - Indicates the uniform resource locator (URL) to use for the specified session; the URL must be less than 1024 characters.
  - This element is optional if the proxy element is used. If a URL is specified in addition to a proxy, then only requests bound for the same end destination will be allowed to use the shared session to the proxy.
- allowRedirects
  - A boolean that specifies whether a request is automatically redirected when the server sends the Location: header as part of the HTTP header. When this field is set to false, the Location: header is not automatically followed.
  - Note: When set to true and automatic redirection is turned on, the HTTP client request will follow redirects to an endpoint that was not originally specified in the HTTP shared session file. Understand that some of the benefits of shared session support, such as isolating the session for traffic to a particular, heavily used endpoint, may be lost

Shared Session File Elements

- proxy
  - Indicates the proxy server information to use for the specified session. The following elements are children of this element: proxyPort and proxyUserPassword. The following attribute is required for this element: name. All proxy information is optional if a URL is being used.
  - This element is optional if a URL is specified. It allows multiple shared sessions to the same proxy each with a different end destination.
  - name
    - Indicates the host name or IP address, in dotted-decimal notation, of the proxy server. The maximum length of this string is 256 bytes.
  - proxyPort
    - An integer that indicates which proxy server port to connect to. This is an integer in the range of 1 to 65535.
  - proxyUserPassword
    - Indicates the user name and password to use when logging in to the HTTP proxy server. The user name and password must be separated by a colon (:) (for example, username:password). The maximum length of this string is 64 bytes.

How Do I Use A Shared HTTP Session?

- **Support for this feature will also be built into the existing HTTP Client APIs**
  - tpf_httpPerform
  - tpf_httpPerform1

What Happens When I Use A Shared HTTP Session?

1. **The request is sent to the HTTP client daemon which does one of the following:**
   - Searches for an existing and available shared HTTP connection to the requested remote destination
   - If all existing sessions are in use an HTTP session with a persistent connection to the remote destination is established (up to maxSessions)
2. **The daemon submits the HTTP request on behalf of the application**
3. **Control is return to the application based on whether the request was synchronous or asynchronous**

What Changes Are Needed To My Existing HTTP Client Applications To Use Shared HTTP Sessions?

- **NONE**

What is the HTTP Client Daemon?

- **A long running set of threaded processes that will act as a “liaison” for asynchronous and shared session requests**
- **The daemon is loosely modeled after the shared SSL daemon**

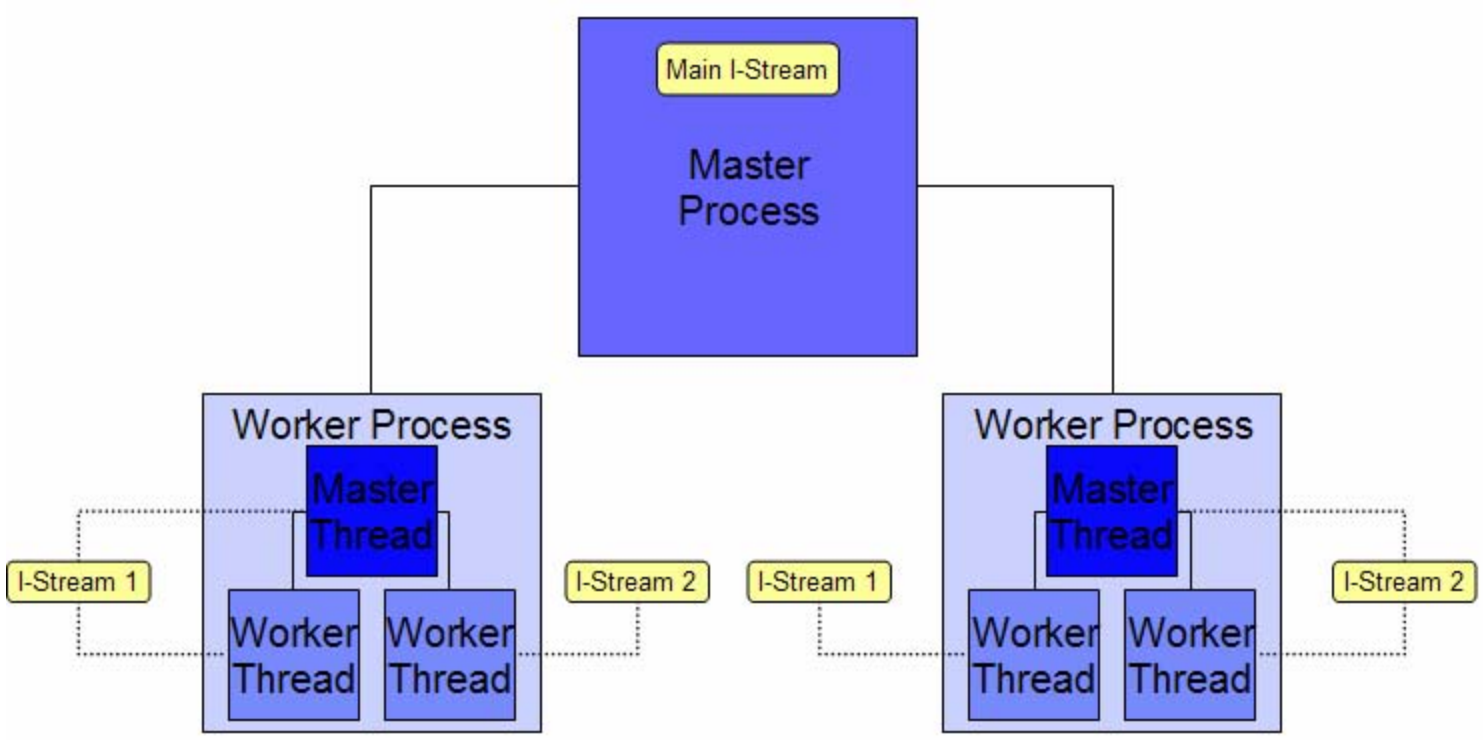
Why Is a Daemon Needed?

- In order to implement both asynchronous and shared session support at least one extra ECB is needed in addition to the application ECB
- For asynchronous support an ECB is needed to wait on and process the HTTP response. This is due to the use of open source libCURL as the underlying HTTP protocol
- For shared session support a long running ECB is needed to own the persistent HTTP connections

How Is The Daemon Configured?

- **One master process**
  - Monitors the worker processes
  - Main I-stream
- **User specified number of worker processes**
  - Manages the worker load across thread
  - Manages ECB heap usage
  - Balanced across I-streams
- **System controlled number of threads per process**
  - Performs the actual HTTP (CURL) request
  - One master thread per process
  - One worker thread for each I-stream

How Is The Daemon Configured?



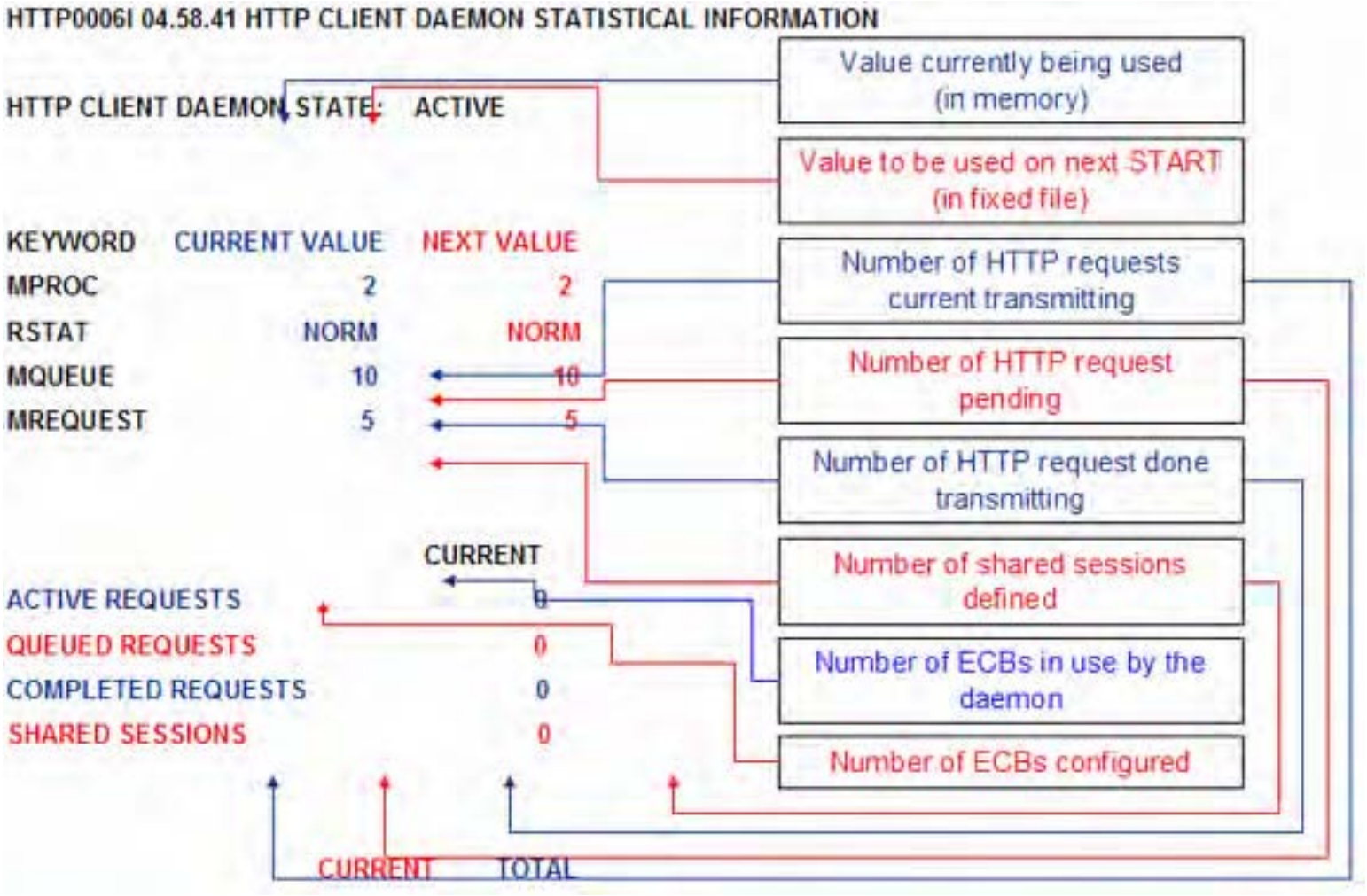
How Can I Configure The Daemon?

- **ZHTTP MODIFY**
- **This command allows the following settings to be adjusted:**
  - The number of daemon worker processes
  - The maximum number of concurrent HTTP requests per thread
  - The maximum queue length for the daemon

How Can I Monitor The Daemon?

- **ZHTTP DISPLAY**
- **The following information about the daemon can be displayed:**
  - Current configuration settings
  - Number of ECBs in use
  - Number of HTTP requests
    - Queued
    - Active
    - Completed
  - Number of shared sessions

AAES0008I 00 ==> ZHTTP DISPLAY



How Can I Monitor The Daemon?

- **ZHTTP DISPLAY SHARED - xxx**
  - ALL – Display information for all shared sessions
  - FAILED – Displays information for session shared that have encounter an error
  - Specific destination – Displays information for the specified shared session

How Can I Monitor The Daemon?

- **ZHTTP DISPLAY SHARED – xxx**
  - URL
  - Proxy
  - Initial/Constant number of connections
  - Maximum number of connections
  - Maximum queue length
  - Current queue length
  - HTTP response code for last request
  - Error code for last request

How Can I Control The Daemon?

- **The daemon can be started, stopped or recycled using the following commands**
  - ZHTTP START
  - ZHTTP STOP
    - Active HTTP request will be allowed to complete
    - Queued HTTP request will give an error to the application
  - ZHTTP RECYCLE
    - A new version on the HTTP client daemon is started
    - HTTP request that are active for the old HTTP client daemon will be allowed to complete
    - HTTP request that are queued for the old HTTP client daemon will be reassigned to the new version of the HTTP client daemon
    - All new HTTP requests will be assigned to the new version of the HTTP client daemon
- **Additional controls include:**
  - Whether or not the daemon should be restarted across an IPL/Cycle-up
  - The state the system should be in when the restart occurs

How Can I Further Customize The Daemon?

- **Daemon Process Selection User Exit**
  - Ability to specify which daemon worker process should handle an HTTP request to a specific destination
  - Interface
    - uhsp_httpDaemonSelectProc(int procCnt,
      BOOL procArray[],
      t_httpOpts httpOptions)
  - Return Values
    - 0 : Normal Return
    - -1: Error Return, this tells the HTTP client daemon not to perform the request (setting errno to ETPFHTTP_UHSP)

What About Soap Consumer Support?

- **Sub-second timeout**
  - Selectable via an existing API (tpf_soapInvokeService)
  - Enabled for the following transports:
    - WebSphere MQ
    - HTTP
    - TPF – to – TPF
- **True asynchronous response handling for SOAP over HTTP**
  - Selectable via an API (tpf_soapInvokeService)

What Changes Are Needed To My Existing SOAP Consumer Applications To Use a TRUE Asynchronous HTTP Transport?

- **NONE**

Putting It All Together

1. Define the number of HTTP client daemon worker processes
   - ZHTTP MODIFY MPROC-x
2. Define the maximum queue length for the daemon as a whole
   - ZHTTP MODIFY MQUEUE-x
3. Define the number of concurrent HTTP requests per thread
   - ZHTTP MODIFY MREQUEST-x
4. Start the HTTP client daemon
   - ZHTTP START

Putting It All Together

SOAP Consumer Application

```
#include <tpf/c_soapc.h>

t_soapHandle    soapHandle;
t_soapInv       soapInvParms;
.
.
.
soapInvParms.invocParmsVerison = SOAPC_VERSION_2;
soapInvParms.timeout = 10;
soapInvParms.timeoutInterval = SOAPC_MILLI;
.
.
.
tpf_soapInvokeService(soapHandle,&soapInvParms,NULL);
```

No application updates are needed for asynchronous HTTP!!

Putting It All Together

Calling Application

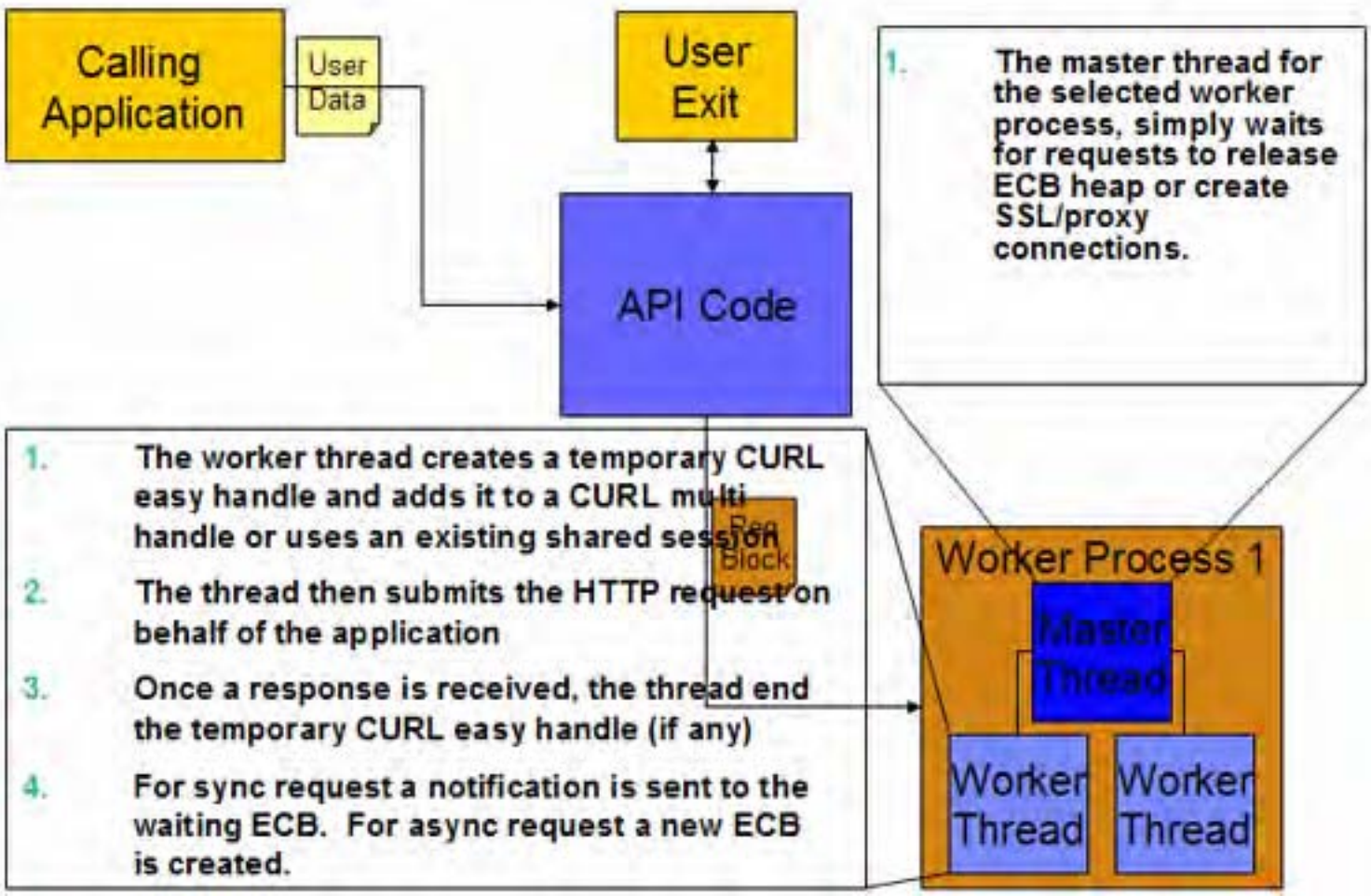
```
#include <tpf/c_httpc.h>
t_httpHandle  httpHandle;
t_httpConn    httpConnParms = HTTPC_CONNECT_VERSION_1;
t_httpRequest httpReqParms  = HTTPC_REQUEST_VERSION_2;

httpReqParms.responseType = HTTPC_ASYNC;
httpReqParms.asyncProgram = "CCHT";
httpReqParms.asyncData = &someStruct;
httpReqParms.asyncDataLength = sizeof(someStruct);
httpReqParms.asyncIS = 0;
httpReqParms.requestTimeout = 10;
httpReqParms.requestTimeoutInterval = HTTPC_MILLI;

httpHandle = tpf_httpInitHandle(&httpConnParms);
tpf_httpPeform(httpHandle,&httpReqParms,NULL);
tpf_httpEnd(httpHandle);

tpf_httpPeform1(&httpConnParms,&httpReqParms,NULL);
```

Putting It All Together

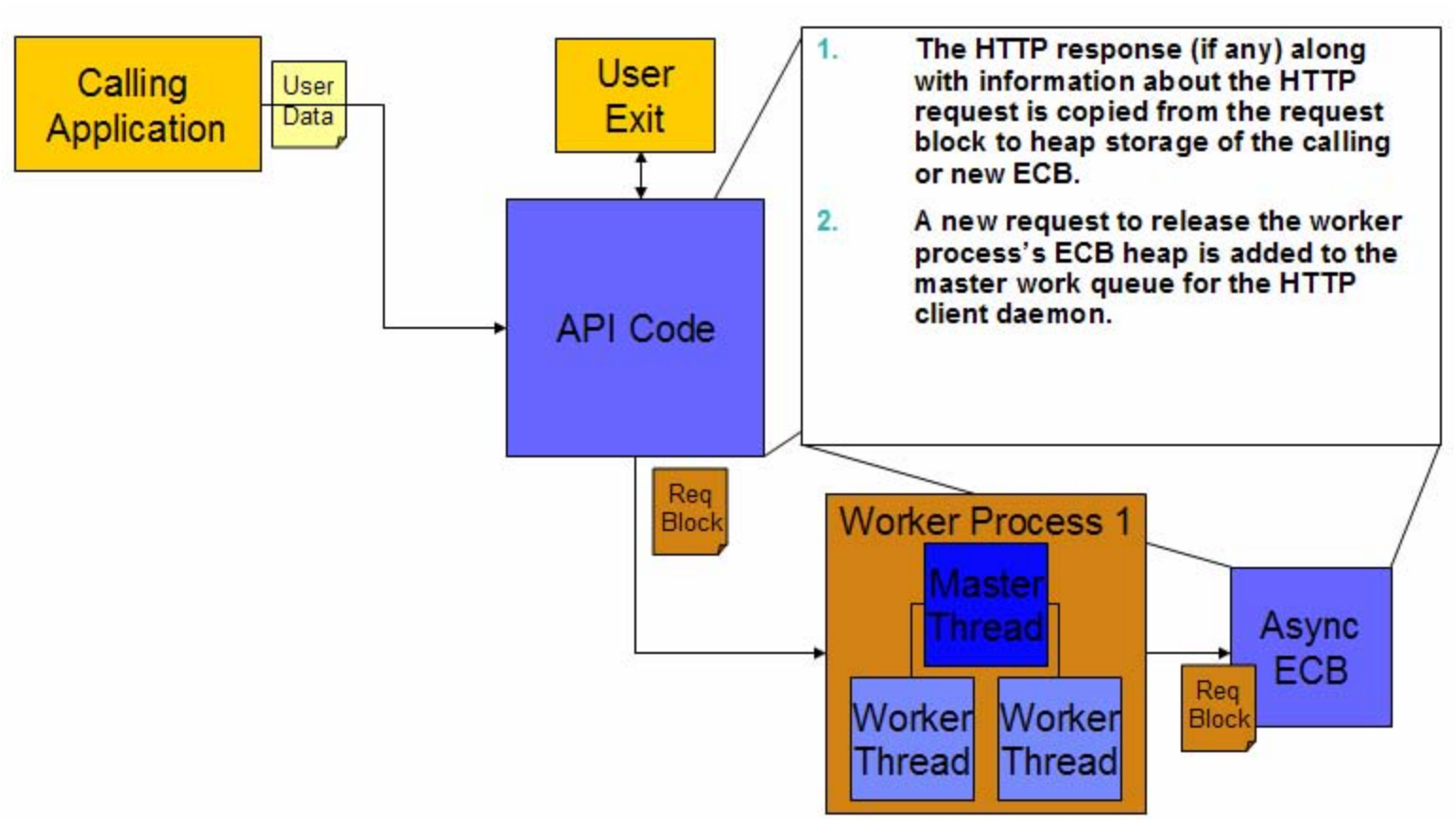
```
#include <tpf/c_httpc.h>
int uhsp_httpDaemonSelectProc(int procCount,
                              BOOL procArray[],
                              t_httpOpts httpOptions)
{
   int rc = TPF_UHSP_OK, i;

   if(strcmp(httpOptions.url,"some_destination" == 0)
   {
      if(procArray[0] == FALSE)
         rc = TPF_UHSP_ERROR;
      else
      {
         for( i = 1; i <procCount; i++)
         {
            procArray[i] = FALSE;
         }
      }
   }

   return(rc);
}
```

1. A check is made to ensure the HTTP client daemon is active.
2. The total queue length of the HTTP client daemon is checked against the user defined value (MQUEUE).
3. Search for a matching shared session
4. A list of available worker processes is created.
5. The process selection user exit is then called.
6. From the worker processes still marked as available, the API code selects the worker thread with the smallest queue.
7. A request block is allocated from system heap.
8. The HTTP client handle and user data including the HTTP request, HTTP headers and async data (if any) is copied into the request block.
9. The request block is placed on the queue of the selected worker thread
10. ECB is suspended for sync request using a shared session or control is returned to the application for async requests

Putting It All Together

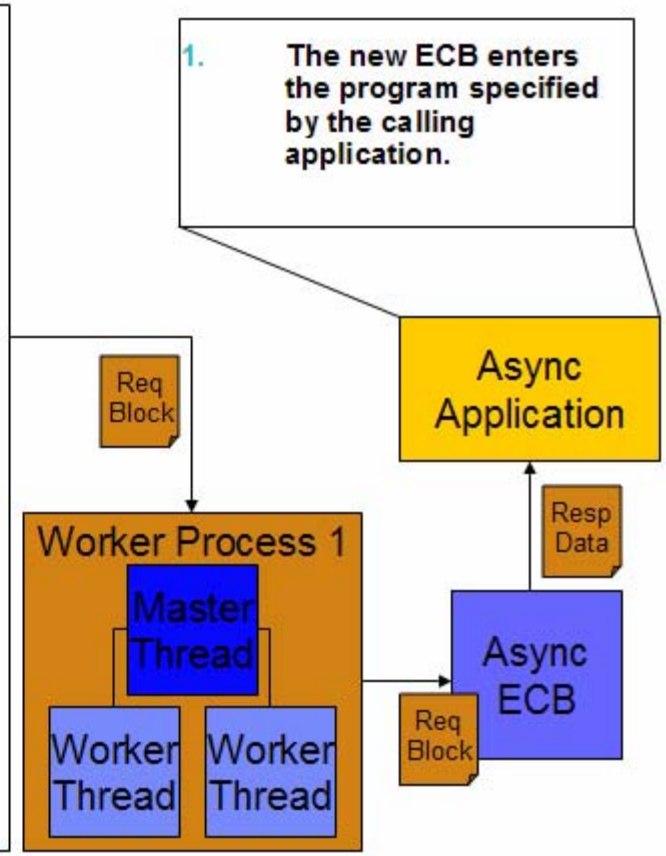


Putting It All Together



Putting It All Together

```
#include <tpf/c_httpc.h>
Void (t_httpAsyncParms *asyncParms)
{
   t_httpConn    httpConnParms = HTTPC_CONNECT_VERSION_1;
   t_httpRequest httpReqParms  = HTTPC_REQUEST_VERSION_2;
   t_httpResponse *httpResp = NULL;

    if(asyncParms->httpReturnCode == TPF_HTTPC_ERROR)
    {
        // All the data needed to resend the request is available
        if (errno ==ETPFHTTP_TIMEOUT)
        {
            httpReqParms.url = &(asyncParms->httpOpts->url);
            httpReqParms.data = asyncParms->data;
            httpReqParms.headerList = asyncParms->headerList;
            httpReqParms.headerNum = asyncParms->headerNum;
            tpf_httpPerform1(&httpConnParms,&httpReqParms,
                             &httpResp);
        }
    }
    else
    {
        //Process the response
    }
   free(asyncParms);
    exit(0);
}
```

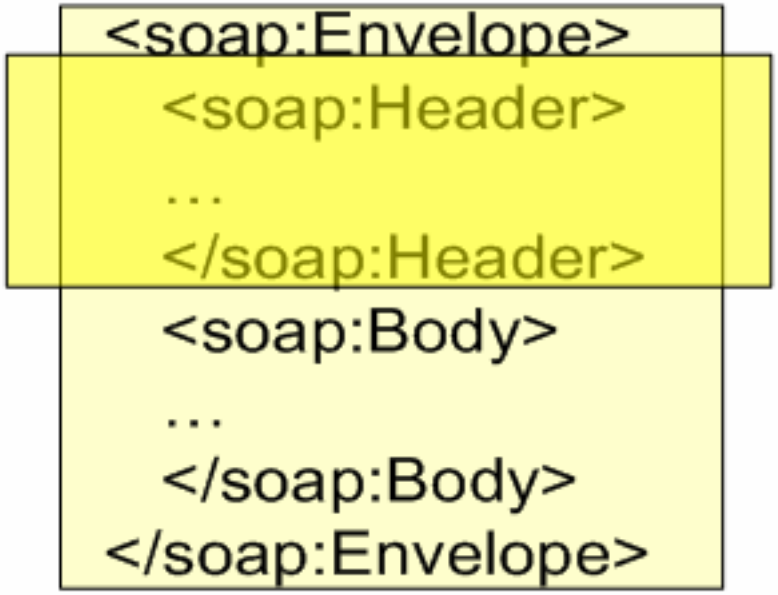


Agenda

- HTTP Client Enhancement
- **SOAP Message Handler Enhancement**
- XML Encryption Support

SOAP Message Handlers

- SOAP message handlers provide a single, reusable interface for extending one or more consumer Web services
  - Standard SOAP features:
    - WS-Security
    - WS-Addressing
    - WS-*
  - User-specific functionality:
    - Logging
    - Billing
- SOAP message handlers perform operations that involve SOAP Headers
  - The SOAP specifications allow for a great deal of extensibility by using the “Header” section of SOAP message



SOAP Message Handlers

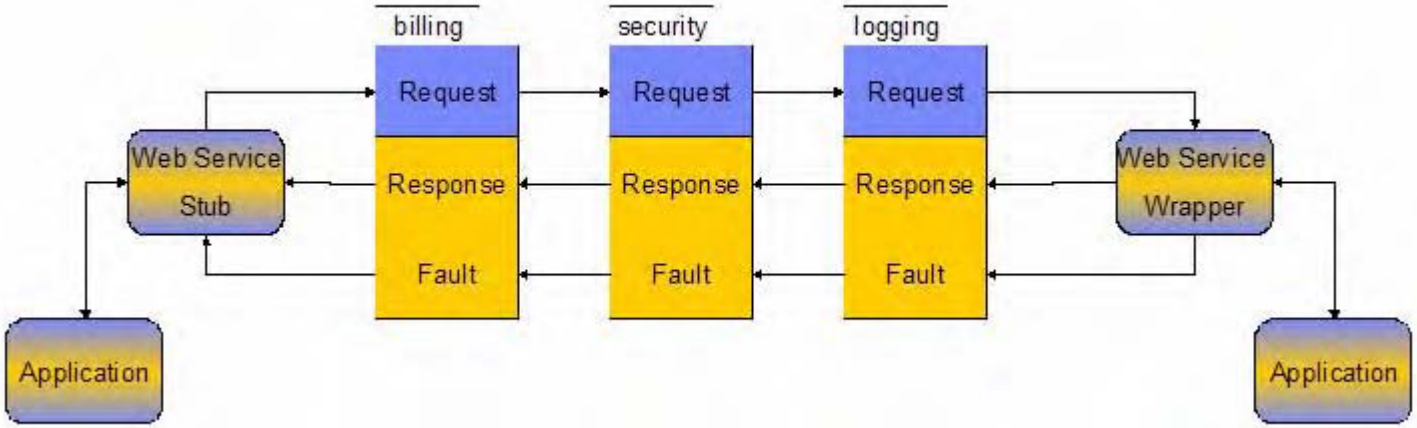
- Each consumer Web service that requires a SOAP message handler to be called must include the message handler's name in the SOAPMessageHandlerChain element of its own deployment descriptor
- Consumer and provider Web Services can share SOAP message handlers
- Creating SOAP message handlers for use on z/TPF requires 2 artifacts:
  - *SOAP message handler deployment descriptor:* defines the runtime characteristics of the SOAP message handler
  - *SOAP message handler program:* performs necessary processing using the outbound SOAP request and the inbound SOAP response

SOAP Message Handlers

- SOAP message handlers are invoked by the z/TPF SOAP provider and consumer support in each of the following situations:
  - **SOAP request**
    - Called in the order they are listed in the Web Service's SOAP message handler chain after calling the Web Service stub or before calling the Web Service wrapper
  - **SOAP response**
    - Called in the reverse order they are listed in the Web Service's SOAP message handler chain before calling the Web Service stub or after returning from the Web Service Wrapper
  - **SOAP fault**
    - Called in the reverse order they are listed in the Web Service's SOAP message handler chain
- SOAP message handlers are not called for SOAP consumer support when the application builds its own XML document.

SOAP Message Handlers

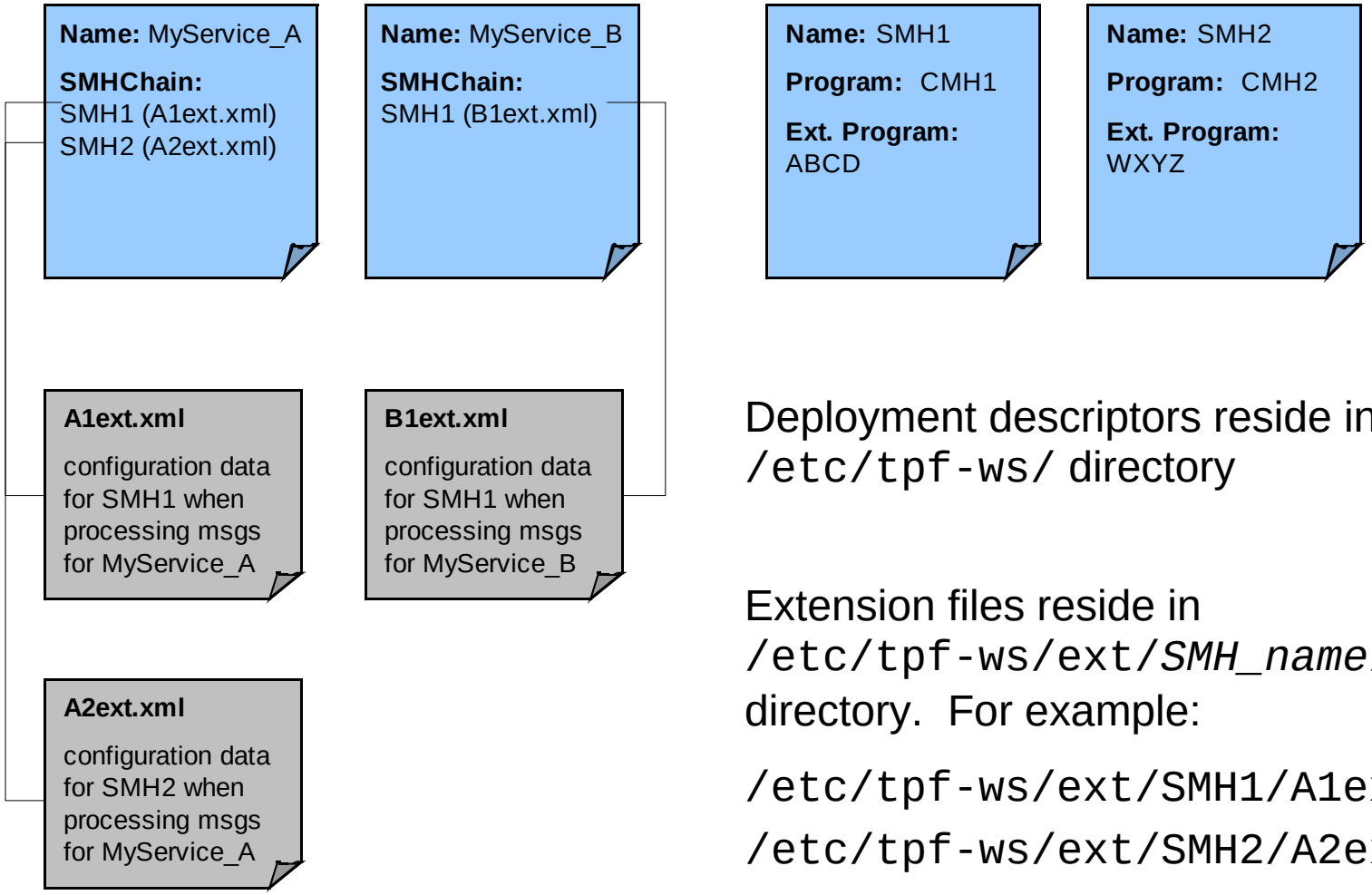
- Flow of the SOAP message handler chain



SOAP Message Handlers – Extension

- **SOAP message handler extension file**
  - SOAP message handlers may require configuration information that is specific to a particular Web service
  - Extension files provide a simple, reusable format for expressing that configuration information and associating it with a given Web service
- **SOAP message handler extension program**
  - This program receives control specifically to perform actions with a SOAP message handler extension file
  - Called during: ZWSAT DEPLOY, UNDEPLOY, REMOVE, and DISPLAY

SOAP Message Handlers – Extension files



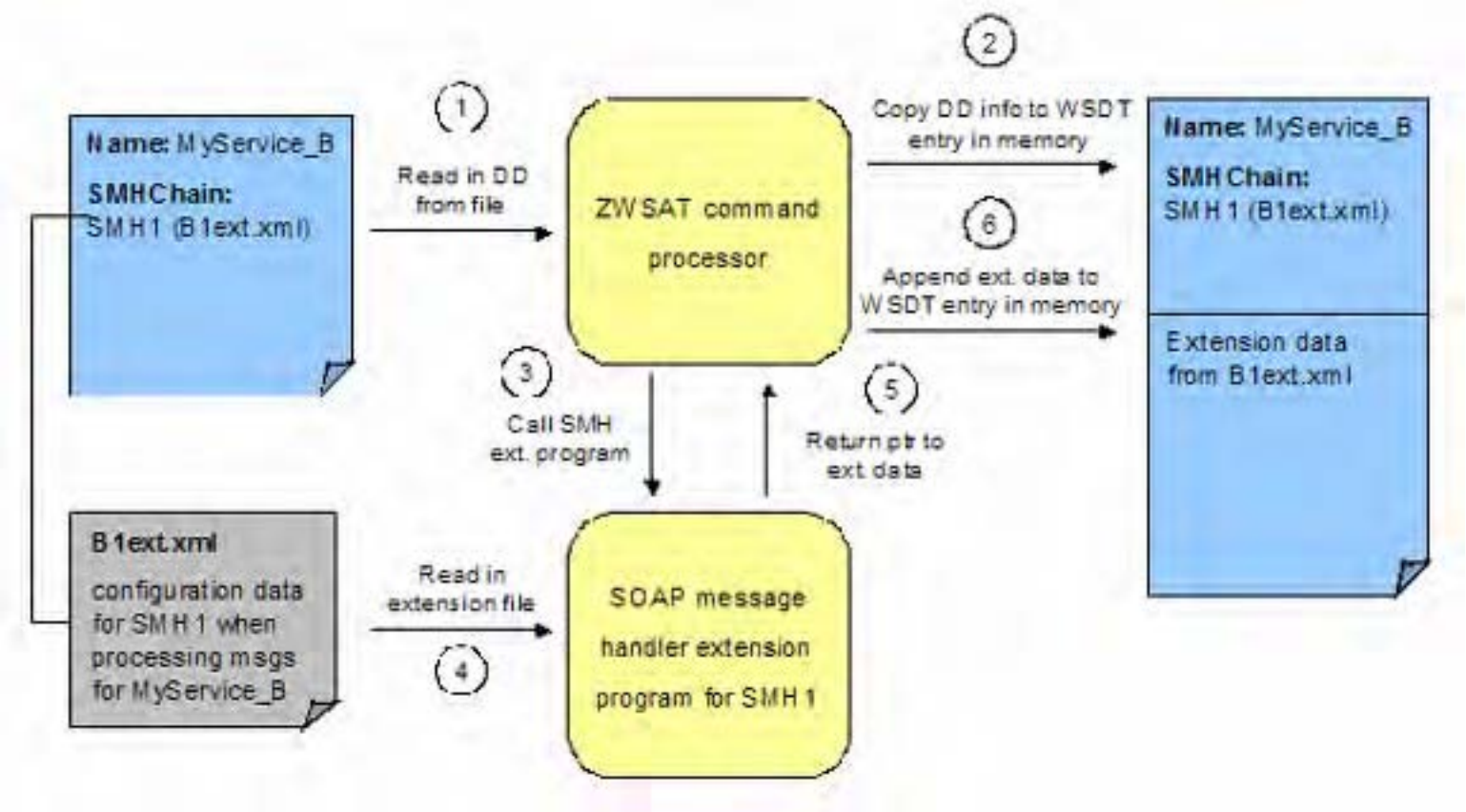
Deployment descriptors reside in `/etc/tpf-ws/` directory

Extension files reside in `/etc/tpf-ws/ext/`*`SMH_name/`* directory. For example:

`/etc/tpf-ws/ext/SMH1/A1ext.xml`

`/etc/tpf-ws/ext/SMH2/A2ext.xml`

SOAP Message Handlers – Extension data



Once the extension data is saved in a Web service's WSDT entry, it will be available to the SOAP message handler via helper functions used to interact with the `tpfSoapMsgCtx` structure while processing Web service messages

Agenda

- HTTP Client Enhancement
- SOAP Message Handlers Enhancement
- **XML Encryption Support**

Web services security (WS-Security)

- **WS-Security is made up of a collection of specifications:**
  - WS-Security Core Specification 1.1 (link)
  - Username Token Profile 1.1 (link)
  - X.509 Token Profile 1.1 (link)
  - SAML Token Profile 1.1 (link)
  - Kerberos Token Profile 1.1 (link)
  - Rights Expression Language (REL) Token Profile 1.1 (link)
  - SOAP with Attachments (SwA) Profile 1.1 (link)
- **WS-Security also builds upon other specifications:**
  - XML encryption (XMLENC) (link)
  - XML digital signatures (XMLDSIG) (link)
  - Canonical XML Version 1.0 (XMLC14N) (link)
  - Exclusive XML Canonicalization Version 1.0 (EXCC14N) (link)
- **Web Services Interoperability Organization (WS-I) clarifies WS-Security:**
  - Basic Security Profile 1.0 (link)

XML encryption

- **Open standard developed by the World Wide Web Consortium (W3C)**
- **Standard specifies the following:**
  - Steps to encrypt data
  - Steps to decrypt encrypted data
  - Syntax to represent XML encrypted data, the information used to decrypt the data, and a list of encryption algorithms supported
- **Data being encrypted/decrypted in an XML document may be either:**
  - An XML element (including child elements if any)
  - XML element content (which may be character data, or all child elements)
- **Result of encrypting data is an `<EncryptedData>` element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data**
  - When encrypting an XML element or element content the `<EncryptedData>` element replaces the element or content in the encrypted version of the XML document

WS-Security usage of XML encryption standard

- **WS-Security defines a `<wsse:Security>` header block where all security related information should be inserted, including information about message encryption**
- **WS-Security Core Specification (WSSE) allows for encryption of any combination of body blocks, header blocks, and any of these sub-structures**
  - When a producer or active-intermediary encrypts portion(s) of a SOAP message it must prepend a sub-element to the `<wsse:Security>` header block
  - The sub-element MUST contain the information necessary for the recipient to identify the portions of the message that it is able to decrypt
    - For SOAP 1.1 z/TPF will only process data if the 'actor' attribute is not specified.
    - For SOAP 1.2 z/TPF will only process data with 'role' specified as ultimateReceive
- **WS-Security also defines a new element called `<wsse11:EncryptedHeader>` for containing encrypted header blocks (similar to `<EncryptedData>` element of XMLENC)**

Identifying encrypted data in SOAP messages

- **`<ReferenceList>` element usage in `<wsse:Security>` header block**
  - May be used to create a manifest of encrypted portions within the SOAP envelope
  - All `<EncryptedData>` elements created should be listed in `<DataReference>` elements inside one or more `<ReferenceList>` elements

```
<S11:Envelope xmlns:S11="..."
  xmlns:wsse="...
      xmlns:wsu="..." xmlns:ds="..."
  xmlns:xenc="...">
  <S11:Header>
     <wsse:Security>
        <xenc:ReferenceList>
           <xenc:DataReference
                 URI="#bodyID"/>
        </xenc:ReferenceList>
     </wsse:Security>
  </S11:Header>
  <S11:Body>
     <xenc:EncryptedData Id="bodyID">
        <ds:KeyInfo>
           <ds:KeyName>
           CN=Hiroshi Maruyama,C=JP
           </ds:KeyName>
        </ds:KeyInfo>
        <xenc:CipherData>
           <xenc:CipherValue>
           ...
           </xenc:CipherValue>
        </xenc:CipherData>
     </xenc:EncryptedData>
  </S11:Body>

</S11:Envelope>
```

Identifying symmetric keys in SOAP messages

- **`<KeyInfo>` element usage in `<wsse:EncryptedData>` element**
  - Specifies the alias of the key used to encrypt the data
- **`<CipherValue>` element usage in `<CipherData>`**
  - Specifies the actual encrypted data

```
<S11:Envelope xmlns:S11="..."
  xmlns:wsse="...
      xmlns:wsu="..." xmlns:ds="..."
  xmlns:xenc="...">
  <S11:Header>
     <wsse:Security>
        <xenc:ReferenceList>
           <xenc:DataReference
                 URI="#bodyID"/>
        </xenc:ReferenceList>
     </wsse:Security>
  </S11:Header>
  <S11:Body>
     <xenc:EncryptedData Id="bodyID">
        <ds:KeyInfo>
           <ds:KeyName>
           CN=Hiroshi Maruyama,C=JP
           </ds:KeyName>
        </ds:KeyInfo>
        <xenc:CipherData>
           <xenc:CipherValue>
           ...
           </xenc:CipherValue>
        </xenc:CipherData>
     </xenc:EncryptedData>
  </S11:Body>

</S11:Envelope>
```

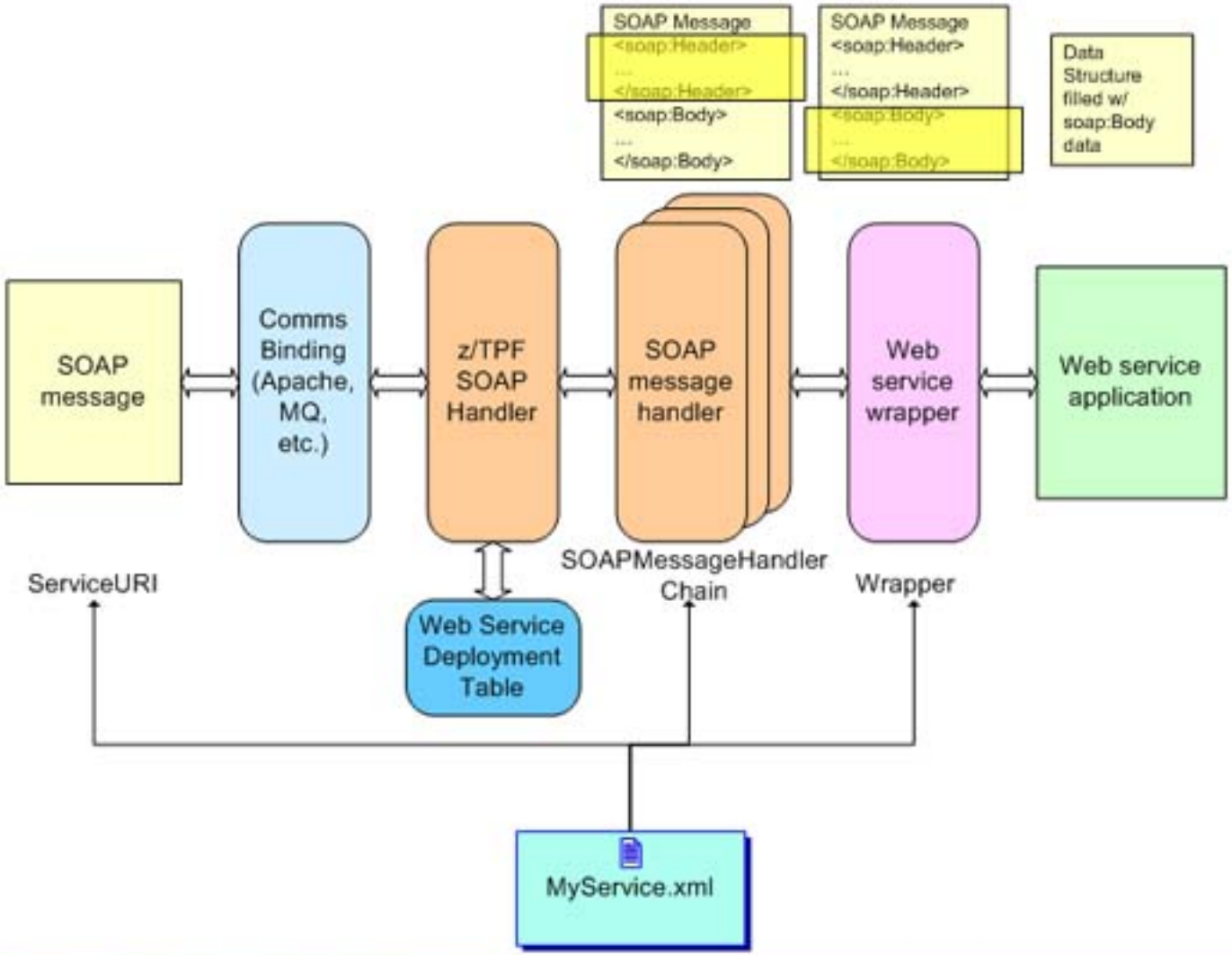
Additional information about encryption with SOAP

- **`<Envelope>`, `<Header>`, and `<Body>` elements must not be encrypted, although child elements may be encrypted**
- **Multiple steps of encryption may be added in a single `<wsse:Security>` header block if they are targeted for the same recipient**

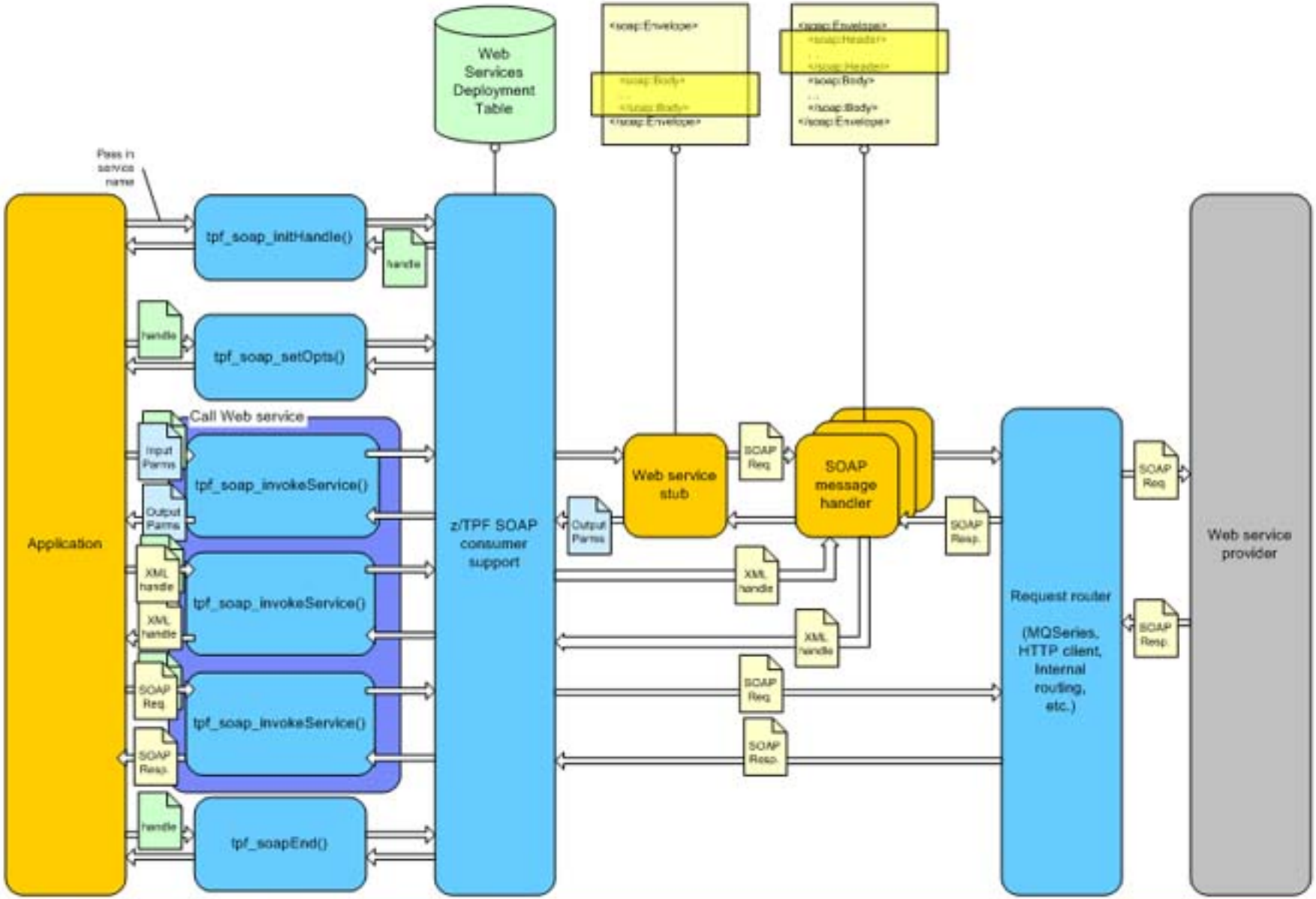
WS-Security SOAP message handler

- **Supported with both SOAP provider and consumer Web services.**
- **Decrypts inbound SOAP requests, responses and faults before passing them to the Web service wrappers/stubs**
  - Protected sensitive data
    - Non-displayable storage for all decrypted information
    - Mechanism to specify sensitive data fields in unencrypted SOAP message
- **Encrypts outbound SOAP requests, responses and faults before passing them to the communications binding/handlers.**

Application flow of SOAP Provider Support



Application Flow for SOAP Consumer Support



WS-Security SOAP message handler for z/TPF

WS-Security Extension File

- **Outflow and Faultflow**
  - Which fields need to be encrypted
    - Identified by a subset of XPath
  - Which keys need to be used
    - z/TPF keystore
    - Specified in user exit
  - How keys are indentified
    - Common alias specified in the message
- **No inflow instructions are required since information about the encrypted data is passed in the SOAP message itself**

How to match keys used on both sides

- **Key alias on z/TPF**
  - User exit to obtain the client ID
  - User exit to map alias
    - To the key name in the z/TPF keystore
    - To actual value of the key
- **Away from z/TPF**
  - Maintain a mapping between actual key names and key aliases specified in SOAP messages
    - Similar to existing SOAP engines, like Apache Axis2

How to enable this support

- **Deploy WS-Security SOAP message handler**
- **For each Web service that requires WS-Security**
  - Create an configuration file with encryption parameters
  - Add WS-Security to SOAP message handler chain and specify the configuration file name
  - Redeploy the service

What Changes Are Needed To My Existing Web Service Applications To Use XML Encryption?

- **NONE**

Availability

- **All enhancement contain in the presentation are available in z/TPF PUT7**
- **HTTP Client Enhancements**
  - APAR 34208
- **libCURL Update**
  - APAR 37296
- **SOAP Message Handler Enhancement**
  - APAR 37735
- **XML Encryption Support**
  - APAR 37735

Trademarks

- **IBM and Websphere ® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.**
- **Other company, product, or service names may be trademarks or service marks of others**
- **Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.**
- **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.**