iMatix Corporation sprl

13/15 rue des Ateliers, 1080 Bruxelles, Belgium
Tel.+32-75-235-984, fax.+32-2-414-8404
http://www.imatix.com
info@imatix.com

Technical White Paper

The Web Transaction Protocol

This document describes the Web Transaction Protocol version 1.0 (WTP/1.0), a protocol that adds transaction processing functionality to HTTP ('web') servers. WTP is an open protocol that can be implemented in many ways. This document defines a reference implementation, which corresponds to the WTP/1.0 implementation offered in the iMatix ***Xitami*** web server.

Copyright

Version Information

Written: 12 January 1998
Revised: 23 January 2000

Disclaimer

The Web Transaction Protocol (WTP)

Overview

The Web Transaction Protocol (WTP) is a replacement for the common gateway interface (CGI) protocol commonly used to build web application programs. WTP corrects some well-known deficiencies in the CGI protocol, and adds transaction management functions specifically required by large-scale applications.

The main difference between CGI and WTP is that CGI is designed for small stand-alone programs, while WTP is designed for multi-program applications. Both protocols provide a method to generate HTML pages in response to URL requests.

Our target application consists of many hundreds or thousands of programs, linked into large executable units called ***application transaction processes*** (ATPs), typically several megabytes large. This application will serve many users across a IP network, using the HTTP server as a connection point, and HTML as the screen presentation language.

Our fundamental requirements for implementing such a large-scale web-based application are:

- Efficiency: while CGI creates a new process for each URL request, we wish to reuse the same process for multiple requests, in series. This is significantly faster.
- Construction: a CGI program is essentially stand-alone; we wish to be able to build applications out of many programs, each handling one logical HTML page or 'screen'. This permits large, complex applications.
- Session control: the user establishes a logical connection when entering the main HTML page (typically a sign-on screen), and maintains this logical connection across a number of URL requests, until it is terminated or broken. This permits intelligent applications.
- Context management: an application program can maintain information about the user's work in progress. For instance, when the user is scrolling through a list of database records, context permits the application program to correctly handle an action like 'Next'. This permits rich applications.
- Distribution: a realistic application may become too large to handle as one executable unit; we wish to be able to break the application into multiple ATPs, without extra work by the programmer. The WTP manager is responsible for locating a suitable ATP to handle a URL request.
- Load balancing: when particular application functions are heavily used, we want to be able to run more than one instance of the same ATP, either statically (by hand) or dynamically (following the flow and ebb of user activity).
- Stability: a realistic application has programs that crash, loop, or corrupt memory. Such programs may not compromise the stability of the overall application.

We can also note that we wanted a protocol that is easy to use, transparent, portable to any platform, any program language, and any HTTP server.

Why Invent A New Protocol?

We considered, and rejected the CGI, FastCGI, and xxAPI (ISAPI, NSAPI, ASAPI, WSX) server plug-in protocols.

- We have built CGI prototypes which include session control and context management. However, such prototypes are not efficient, and do not allow construction, distribution, or load balancing.
- The xxAPI plug-in protocols require highly-skilled developers, and do not allow distribution, load-balancing, or context management. We do not believe that a xxAPI can provide an efficient model for heavy data processing: when a database operation takes many seconds to complete, the entire web server is blocked during this period. An xxAPI application cannot be guaranteed to be either stable or portable.
- We looked at the FastCGI protocol from OpenMarket (www.openmarket.com); this tackles the issue of efficiency, but not session control, context management, or distribution: a FastCGI application is a single executable unit. We considered adding session control and context management to this protocol (as we did for CGI), but that still leaves the issue of distribution unresolved.
- We looked at various CGI-hybrids for building web applications; most of these are a mixture of CGI combined with a server process: each URL request is passed to a small CGI program that makes a connection to the application server, sends the request, waits for a response, then returns that to the HTTP server. We did not choose such a model for various reasons. Firstly, it still requires a new process for each URL request, which will always be inefficient at high loads. Secondly, it passes all requests to a single server, which must either be multithreaded (i.e. complex) or single-threaded (i.e. slow). Neither of these match our needs. However, it would be possible to implement WTP in this manner, if one did not want to write a HTTP server plug-in (see figure 4).

The design of WTP is, therefore, a combination of these existing web application protocols with a solid transaction processing system that is as powerful as existing mainframe transaction processors. We consider transaction processing to be an essential basis for any realistic large-scale application.

WTP is implemented by a WTP manager program. The WTP manager can be embedded into the HTTP server (for instance the iMatix **Xitami** web server supports WTP directly); it can be built as an xxAPI plug-in; it can even be implemented as a FastCGI program, or as a CGI-hybrid program.

This figure shows WTP support built-in to the web server:

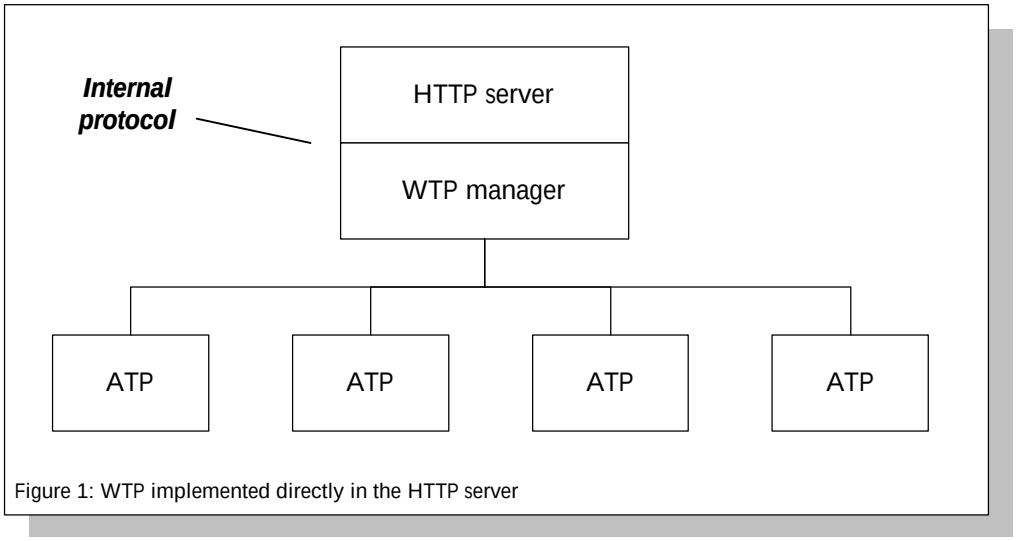

Figure 1: WTP implemented directly in the HTTP server

This figure shows WTP support built as a server plug-in:

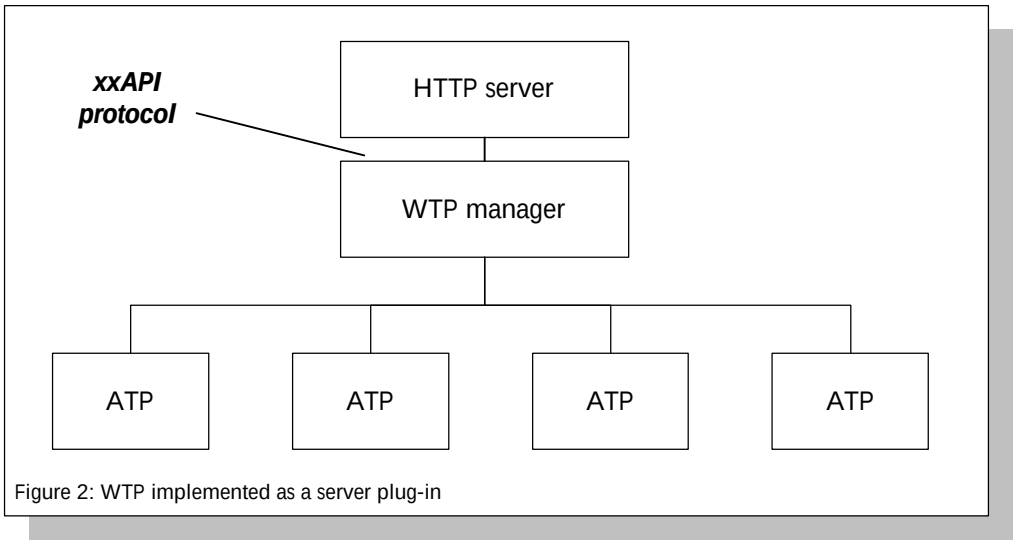

Figure 2: WTP implemented as a server plug-in

This figure shows WTP support built using FastCGI or a similar protocol:

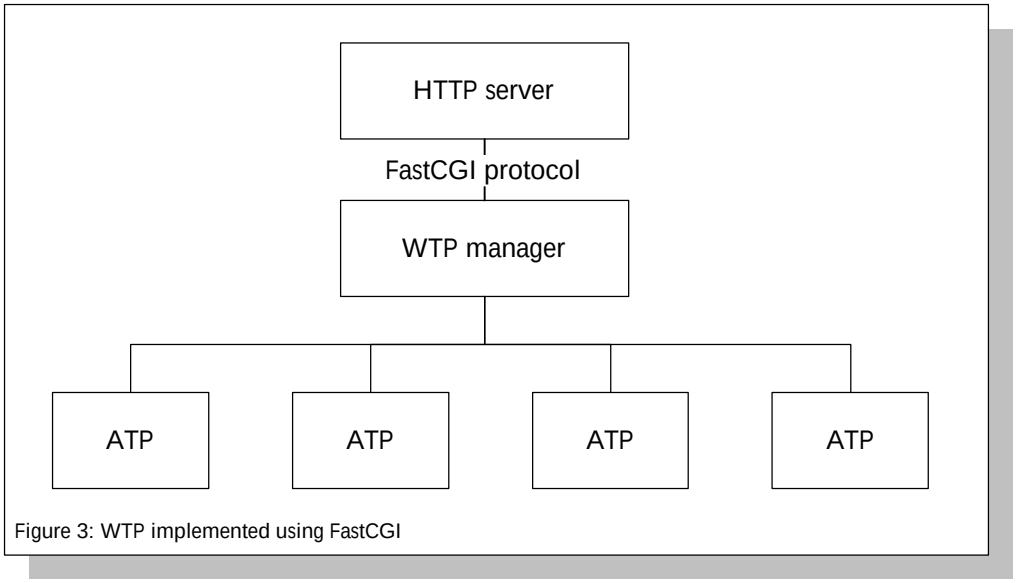

Figure 3: WTP implemented using FastCGI

This figure shows WTP support built as a CGI-hybrid program:

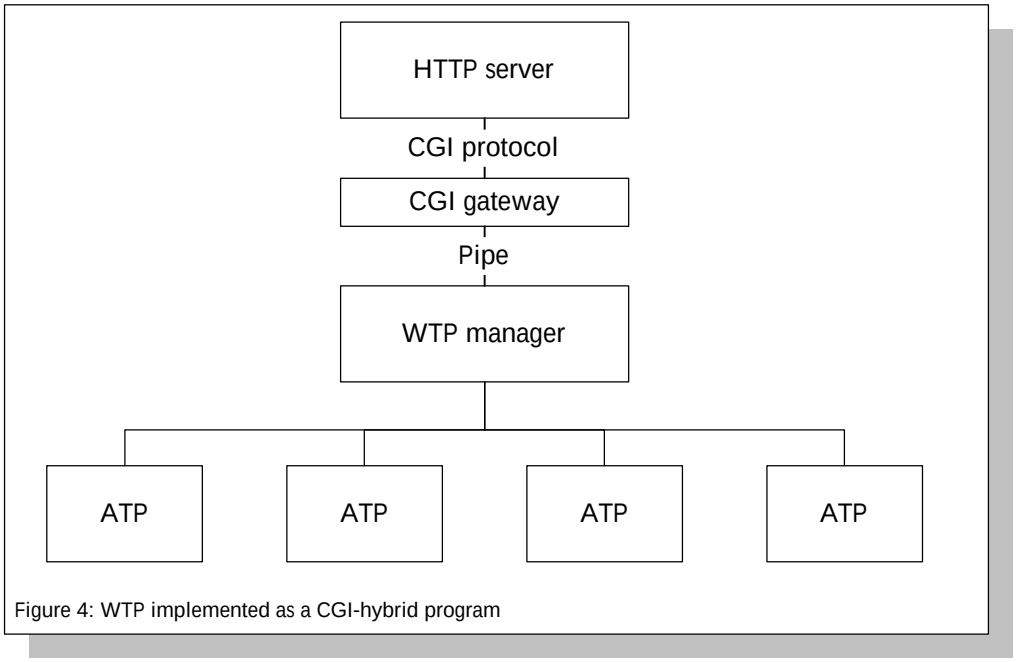

Figure 4: WTP implemented as a CGI-hybrid program

How WTP Works

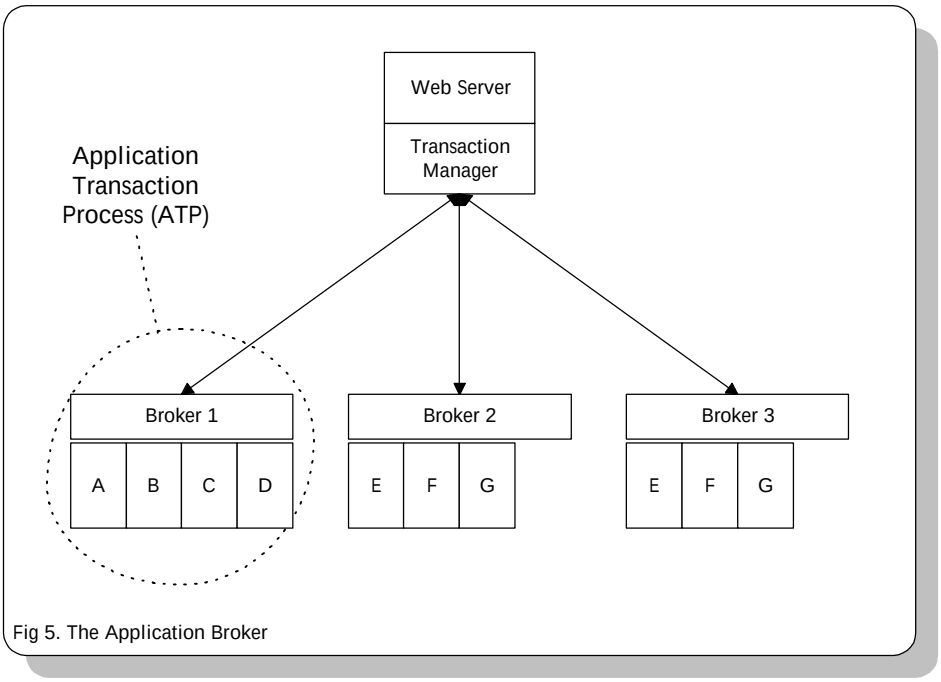

Fig 5. The Application Broker

The WTP manager is responsible for starting, monitoring, and halting ATP processes as required. One WTP manager (there may be several active on a particular host machine) is responsible for handling a set of WTP applications. For instance, one WTP manager could handle the development, test, and production versions of a client control application. Each of these applications can be stopped and started independently.

In practice we would use a separate server for production applications, to ensure the highest possible degree of stability and reliability.

Communications between the WTP manager and ATPs use a 'callback' mechanism as follows: the WTP manager creates a TCP or UPD port that accepts connection requests. When the WTP decides to start the application, it creates one process per ATP. The ATP starts-up, and connects to the WTP manager port. The ATP then 'registers' with the WTP manager, so that the WTP manager knows what work the ATP is able to do. The ATP then waits for requests from the WTP manager: when a request arrives, it handles it and responds with a reply. At any moment the WTP manager can choose to kill the ATP, or create further instances; equally the ATP can handle fatal errors by aborting if necessary.

ATP Initialisation

When the WTP manager starts an ATP, it passes a number of command-line arguments to the ATP main function:

- A VERSION string. This is "WTP/x.x" where 'x.x' is the WTP version number supported by the WTP manager. The ATP main function should check this string and take

appropriate action. For example, if the ATP cannot handle the WTP version, it can write an error message to the stderr stream, and exit.

- A PROTOCOL specifier. This may be 'tcp', 'udp', or 'rdtp', and indicates which protocol the WTP manager can handle on its callback port. TCP is the best-known internet protocol; UDP is a simpler and faster protocol suitable to local connections; RDTP is an experimental protocol being developed by iMatix that combines the speed advantages of UDP with the reliability of TCP.
- A CALLBACK PORT number. This is specified as a string, e.g. "5500".
- A CALLBACK KEY. This is a string, e.g. "P83hXSb8AzyU", that the ATP must supply during connection. The purpose of the callback key is to ensure that only authorised ATPs try to connect to the callback port. The WTP manager generates a unique callback key for each ATP instance that it starts.

Using this information, the ATP connects to the WTP manager (by sending a WTP_CONNECT message), then registers a number of application programs by sending zero or more WTP_REGISTER messages.

Typically we build the callback and registration logic into the ATP main function. We call this the 'broker program'. Broker programs can be written by hand, or generated. The WTP toolkit includes tools to generate these programs, and function libraries to encapsulate much of the necessary work.

WTP Messages

WTP messages use a compact representation aimed at efficiency rather than readability. We did not choose the style of a HTTP message for two reasons. Firstly, HTTP messages are not explicitly sized, so cause difficulties for persistent connections. I.e. the original HTTP protocol assumed that the end of a message was equivalent to the end of a connection. Secondly, HTTP messages are quite verbose, an overhead that we wanted to avoid.

This is the format of a WTP message:

```
[message size]      4 bytes, in network order (hi to lo)
[message type]      1 byte, defining the message type
[message body]      zero or more fields
```

The message size specifies the size of the message excluding the two size bytes. The wtpdefn.h file defines a set of constants for C programs that use WTP. The message body consists of zero or more fields, implicitly defined by the type of message. Fields can be any of these types:

Field type:	Has this meaning:
byte	A 1-byte value
dbyte	A 2-byte value, in network byte order
qbyte	A 4-byte value, in network byte order

string	A null-terminated string
block	A block of data, specified as a four-byte size field plus a series of bytes. Not null terminated.

These are the WTP messages that an ATP can send to the WTP manager:

WTP_CONNECT	Connects to the WTP manager
WTP_REGISTER	Registers a program
WTP_READY	Signal ready for work
WTP_DISCONNECT	Disconnect from the WTP manager
WTP_ERROR	Request failed
WTP_DONESHOW	End program; show HTML screen
WTP_DONECALL	End program; call new program
WTP_DONERETURN	End program; return to calling program
WTP_DONEEXIT	End program; exit the application
WTP_DONEERROR	End program; there was a fatal error

These are the WTP messages that the WTP manager can send to an ATP:

WTP_DO	Execute some program
WTP_OK	Request suceeded
WTP_ERROR	Request failed
WTP_DISCONNECT	ATP should terminate

All messages are sent on the basis of 'question and response'. Invalid messages get a WTP_ERROR reply with the error code WTP_ERRORINVALID. The WTP manager may try to recover from an invalid message, or may break the connection.

The WTP_CONNECT Message

```
[message type]   byte    WTP_CONNECT
[callback key]   string  As supplied by the WTP manager
[signature]      qbyte   Version identification signature
```

This message must be the first message that an ATP sends to the WTP manager. It establishes the logical connection between the ATP and the WTP manager. The callback key is used by the WTP manager to ensure that only real and valid ATPs can connect. The signature string is a 32-bit value the ATP should generate from its executable file date and time. This is used to allow detection of incompatible or changed ATP executable versions.

The WTP manager responds to a WTP_CONNECT message with a WTP_OK or a WTP_ERROR message, with one of these error codes:

WTP ERRORUNAUTHORISED - An invalid callback key was supplied

WTP ERRORUNEXPECTED - Not allowed at this point

The WTP_REGISTER Message

```
[message type]   byte    WTP_REGISTER
[program name]   string  Name of program
```

```
[is root]        byte    1 if this is the root program, else 0.
```

This message tells the WTP manager which programs that the ATP is able to run. One program generally corresponds to a HTML screen. The ATP sends one WTP_REGISTER message for each program it contains. If no root program is specified, the WTP manager may reject any connection to the application with an appropriate error message. If several root programs are specified, the WTP manager may choose to use the first, the last, or use some other algorithm to decide which root program to launch. Typical applications will specific exactly one root program.

The WTP manager responds to a WTP_REGISTER message with a WTP_OK or a WTP_ERROR message, with one of these error codes:

WTP ERRORUNCONNECTED - WTP CONNECT was not sent, or failed

WTP ERRORUNEXPECTED - Not allowed at this point

The WTP_READY Message

```
[message type]   byte    WTP_READY
```

This message tells the WTP manager that the ATP is ready to accept application program requests. The WTP manager responds to a WTP_CONNECT message with a WTP_OK or a WTP_ERROR message, with one of these error codes:

WTP ERRORUNCONNECTED - WTP CONNECT was not sent, or failed

WTP ERRORUNEXPECTED - Not allowed at this point

The WTP_DISCONNECT Message

```
[message type]   byte    WTP_DISCONNECT
```

This message allows an ATP to terminate the connection to the WTP manager. This message is not strictly needed, since the WTP manager will detect that an ATP has terminated, and handle the disconnection automatically. The WTP manager does not respond to a WTP_DISCONNECT message.

The WTP_OK Message

```
[message type]   byte    WTP_OK
```

This message is sent as a positive response, and never receives a response.

The WTP_ERROR Message

```
[message type]   byte    WTP_ERROR
[error code]     dbyte   Cause of the error, as a numeric code
[error reason]   string  Cause of the error, as a string
```

This message is sent as a negative response, and never receives a response.

The WTP_DO Message

```
[message type]   byte    WTP_DO
[signature]      qbyte   Version identification signature
```

```
[program name]    string  Program to execute
[entry code]      byte    Program entry code
[HTTP URI]        string  URI for use in HTML hyperlinks
[HTTP data]       string  Encoded HTTP query data, if any
[arguments]       block   Program call arguments, if any
[call result]     byte    Call result indicator
[environment]     block   HTTP environment block
[global context] block    Global context block
[local context]   block   Local context block
```

This message asks the ATP to execute a specific program. The entry code can be one of:

WTP DOINIT Initial entry into the program

WTP DOGET Program has to process HTML form data

WTP DOCONTINUE A called program finished its work

The use of the entry code is explained in the section "The WTP Program Model".

The signature is that supplied by the ATP at connection time. The ATP should recalculate the signature, and if it fails to match, return a WTP_ERRORSIGNATURE code.

The HTTP URI must be used by the application programs when they create HTML links in their HTML screens. The URI is encoded to contain a 'session key', i.e. information that the WTP manager needs to identify the session when the user uses an action on the HTML form. The HTTP URI is explained in the section "WTP Session Control".

When the program execution state is WTP_DOGET, the HTTP data string holds the encoded HTTP form or query data. Otherwise this string is empty (a single null byte). The format of this data is explained in the section "HTTP Form Data Encoding". The call arguments block is empty.

When the program state is WTP_DOINIT, the call arguments block holds the arguments supplied by the calling program. If the program being executed is the application root program (i.e. it has no calling program), then the call arguments may be empty, or may contain any 'command line' arguments specified by the user in the URL which invokved the WTP application.

When the program state is WTP_DOCONTINUE, the call arguments block holds the return arguments from the called program, and the call result indicator is set to one of the values listed below.

These are the possible values for the call result indicator:

WTP NOERROR Call succeeded

WTP ERRORNOTFOUND Requested program is not known

WTP ERRORWOULDLOOP Requested program is already active

WTP ERROROVERFLOW Maximum number of active programs reached

The HTTP environment block contains the HTTP header fields and standard CGI variables (like REMOTE_HOST). This block is only supplied to the application root program when it starts, since it is essentially identical for all WTP_DO messages for a session. At other times this block is empty.

The ATP responds to a WTP_DO message with WTP_DONExxxx if there were no problems, or WTP_ERROR if there was a problem, with one of these error codes:

WTP ERRORNOTFOUND The program is not known

WTP ERRORUNAVAILABLE The program is no longer available

WTP ERRORSIGNATURE The ATP signature has changedW

TP ERRORUNEXPECTED Not allowed at this point

The WTP_DONESHOW Message

```
[message type]   byte    WTP_DONESHOW
[HTML data]      string  HTML screen data
[global context] block   Global context block
[local context]  block   Local context block
```

An application program has finished a logical unit of work when it (a) is ready to display a form, (b) wants to calls another application program, or (c) has terminated, either normally, or following some error. In the first of these cases, it returns a WTP_DONESHOW message.

The WTP_DONECALL Message

```
[message type]   byte    WTP_DONECALL
[program name]   string  Program to call
[arguments]      block   Arguments for called program
[global context] block   Global context block
[local context]  block   Local context block
```

This message tells the WTP manager to call a new program. The current program is suspended, and will resume only when the called program sends a WTP_DONERETURN message. No HTML is sent to the user at this point; the WTP manager must locate and start the requested program.

The WTP_DONERETURN Message

```
[message type]   byte    WTP_DONERETURN
[arguments]      block   Arguments back to parent program
[global context] block   Global context block
```

This message tells the WTP manager to return to the previous parent program. If the current program was the root program, this message is treated as a WTP_DONEEXIT message.

The WTP_DONEEXIT Message

```
[message type]   byte    WTP_DONEEXIT
```

This message tells the WTP manager to end the application session.

The WTP_DONEERROR Message

```
[message type]   byte     WTP_DONEERROR
[error reason]   string   Cause of the error, as a string
```

This message tells the WTP manager to end the application session, and show an error message to the user.

The WTP Program Model

The WTP program model enforces a transaction-based model. This was a deliberate design decision: our long experience in building successful large-scale business applications has taught us that this is a good way to build efficient, cheap, and robust applications.

These are the main differences between a 'normal' program and a WTP program:

1. The WTP program must send all its data to the client screen in one operation. Furthermore, this action is fused to the end of the transaction. There is no way for the program to display some data, wait for some input, and so on. This is a model that is well-known to CGI programmers, but less evident to Windows and UNIX programmers. In short, WTP uses the standard HTTP 'thin client' model.

2. WTP transaction ends when the program decides to display its HTML page. At this time, the database transaction (if any) is closed; all outstanding database requests are either committed or rolled-back; any open files are closed, and any temporary memory is released. A WTP transaction cannot remain 'open' while the user inputs data, for several reasons. Firstly, database resources may never be locked for more than a few seconds at most, to avoid deadlocks. Secondly, since WTP permits multiple instances of an ATP for load balancing, any process-specific resources (dynamically-allocated memory, open files,...) cannot be guarantied to be available when the program continues processing after receiving the form.

3. The actions of showing the HTML page, calling another program, or returning to the caller program are formalised and handled by the WTP manager, not the program. Again, this is necessary given the WTP distribution and load-balancing functions.

4. WTP program is invoked in different ways depending on the situation. The WTP_DO message uses WTP_DOINIT when the program is newly activated. It uses WTP_DOGET when the program is re-activated to handle HTTP form data. It uses WTP_DOCONTINUE when the program is re-activated after a called program ended.

5. Similarly, a WTP program must signal its intentions to the WTP manager. It does this by using different messages. WTP_DONESHOW means it wants to display an HTML page. When the user uses some action on the HTML page, the same program is re-activated with a WTP_DOGET entry code. WTP_DONECALL means it wants to call another program. WTP_DONERETURN means it has finished. WTP_DONEEXIT means it has decided to end the user session.

Walkthrough Of A WTP Transaction

Here we show the transactions involved in a typical operation, user sign-on. We show the principal sign-on screen, accept a user sign-on, and show a top-level menu screen. Finally we return to the sign-on screen:

- The WTP manager receives a user URL request for the application. It determines the main program, and sends a WTP_DO + WTP_DOINIT message to the appropriate ATP.
- The main program prepares the sign-on form, clears the user-name and password fields, and returns WTP_DONESHOW.
- The WTP manager - via the web server - displays the HTML page and waits for user input.
- The user enters data into the user-name and password fields, then clicks on the 'Sign-on' button. The web browser now sends the form data back to the web server, which passes it to the WTP manager.
- The WTP manager decodes the form data to extract a session key. Armed with this, it calls the main program once again with WTP_DO + WTP_DOGET.
- The main program decodes the HTTP form data, verifies the user name and password, and if it accepts them, decides to call the top-level menu program. It returns WTP_DONECALL.
- The WTP manager locates the ATP for the top-level menu program, then sends WTP_DO + WTP_DOINIT to the ATP.
- The menu program prepares its screen and returns WTP_DONESHOW.
- The user clicks on the 'Exit' button. The menu program receives the WTP_DO + WTP_DOGET, and returns WTP_DONERETURN.
- The WTP manager now sends a WTP_DO + WTP_DOCONTINUE back to the main program, which eventually replies with a WTP_DONESHOW.

WTP Session Control

The WTK manager is responsible for creating and managing the WTP session. There are many possible ways to do this; the choice of design is transparent for WTP applications; we describe one possible implementation, and our reasons for choosing it.

HTTP is a stateless protocol, but there are a number of ways to add state to a HTTP conversation. One common technique is 'cookies'. These are small strings of data that the server returns with a page. The browser will include these back in any later response. Unfortunately, cookies are often (mis)used as a technique to track user's access to a particular site; as a result many people disable their browsers' cookie functions. Another

technique is to use hidden form fields. These fields are returned with the form when the user clicks on an action. Hidden form fields work well when all actions on a HTML page are implemented as submit buttons. There are cases, however, where this is cosmetically unacceptable. One example is where the user can make a selection from a list of client records. Such a list looks and works much better using hyperlinks. However, browsers do not interpret hyperlinks as form submission actions. (This can be programmed in JavaScript, but painfully, and -- to our knowledge -- only on one version of one browser, and that thanks to a bug.) The last candidate technique is to encode the session information in the URI used in hyperlinks. This requires that at the moment the HTML page is generated, the encoded URI be inserted into hyperlinks, along with other data sufficient to allow the application to use the resulting 'click'. An encoded URI could looks like this: "/wtp/application/?session=XYZ123". However, the WTP manager can choose any suitable encoding it likes, since it is solely responsible for decoding the URI.

The WTP manager supplies a suitable URI each time it sends a WTP_DO message to an ATP. This URI must at least specify the WTP application so that a hyperlink returns correctly to the WTP manager. If the WTP manager implements state using cookies, for instance, it must still supply a valid URI to the ATP.

The WTP URL Format

The format of a WTP application URL is:

```
http://hostname[:port]/wtp/application[?arguments]
```

The application can be specified as one or more levels, e.g.:

```
http://www.imatix.com/wtp/clients/dev/
```

Context Management

The WTP_DO and WTP_DONExxxx messages include two blocks called the 'global context' and 'local context'. The global context block is an area of memory that is shared between all programs in a session. This can be used to store information that is pertinent to the whole session, for instance information about the user. The global context block is initialised as an empty block (size zero) when the session is created. All WTP_DONExxxx messages update the global context block.

The local context block holds information for the current program only. The WTP manager initialises this block when starting a new program (either the root program or following a DONE_CALL). It deletes the block when the program terminates (DONE_RETURN).

HTTP Form Data Encoding

The HTTP form data encoding format (sometimes called 'MIME' encoding) is identical to that provided to CGI programs on their stdin stream or on their command line. The HTTP data consists of a series of encoded 'name=value' pairs, separated by & or ; characters. Each 'name=value' pair is encoded using the following escape mechanism: all characters except alphanumerics and spaces are converted into the 3-byte sequence "%xx" where xx

is the character's hexadecimal value; spaces are replaced by '+'. Line breaks are stored as "%0D%0A", where a 'line break' is any one of: "\n", "\r", "\n\r", or "\r\n". The WTP support libraries provide functions to decode and access such data strings.

A WTP application will typically be driven by HTTP POST operations (in which data from a form is posted) and by HTTP GET operations (typically the result of hyperlinks or direct requests to a page). In general, a POST can only be done through a push-button or image; a GET can be done through a hyperlinked text or image.

With suitable encoding, a GET operation will return data that can be used much as POSTed data. To allow the WTP application to detect that data was provided by GET arguments rather than through a POST, we use the convention that GET argument data starts with '&'. This extra character can be skipped by the HTTP decoding routines.

Support for National Character Sets

The HTTP form data can be encoded using the SGML meta-characters for non-portable national characters. However, the WTP manager will do a reasonable attempt to translate characters where it can. It will do this on output only.