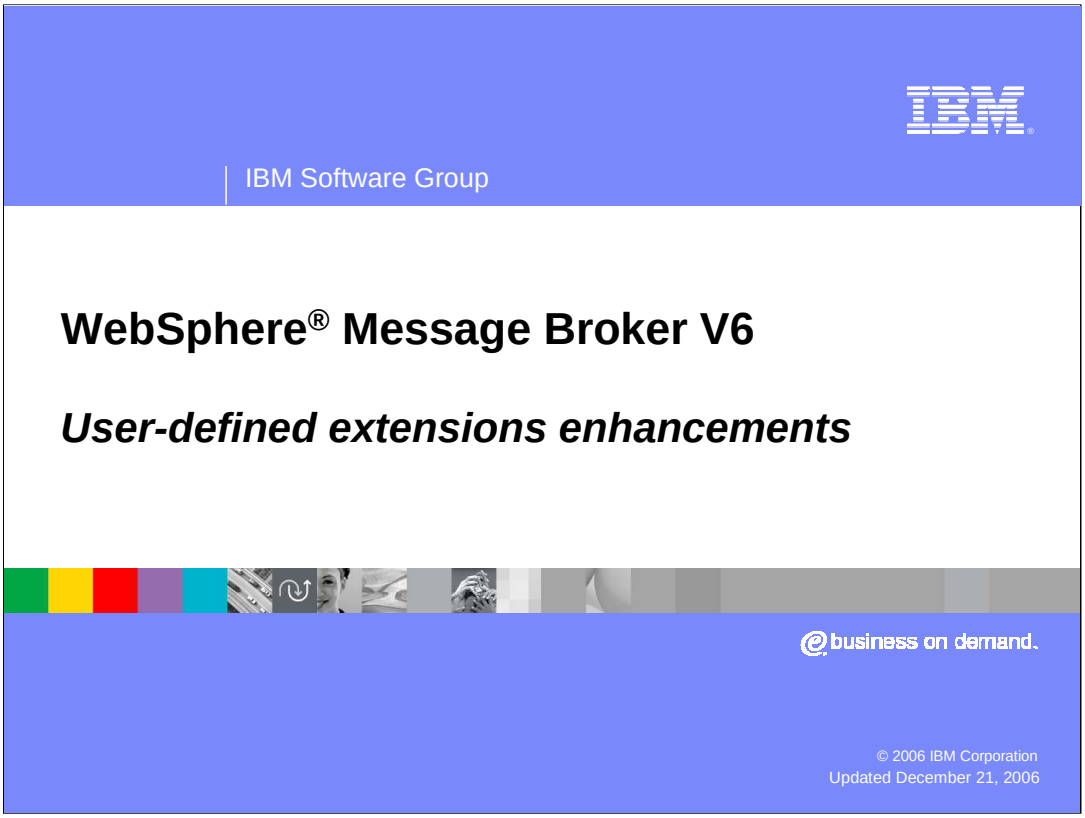


This presentation discusses the enhancements in user-defined extensions implemented in WebSphere Message Broker V6.

Agenda

- User-defined extensions overview
- User-defined extensions enhancements
- Summary and References

The agenda for this presentation includes an overview of user-defined extensions and enhancements.

Section

User-defined extensions overview

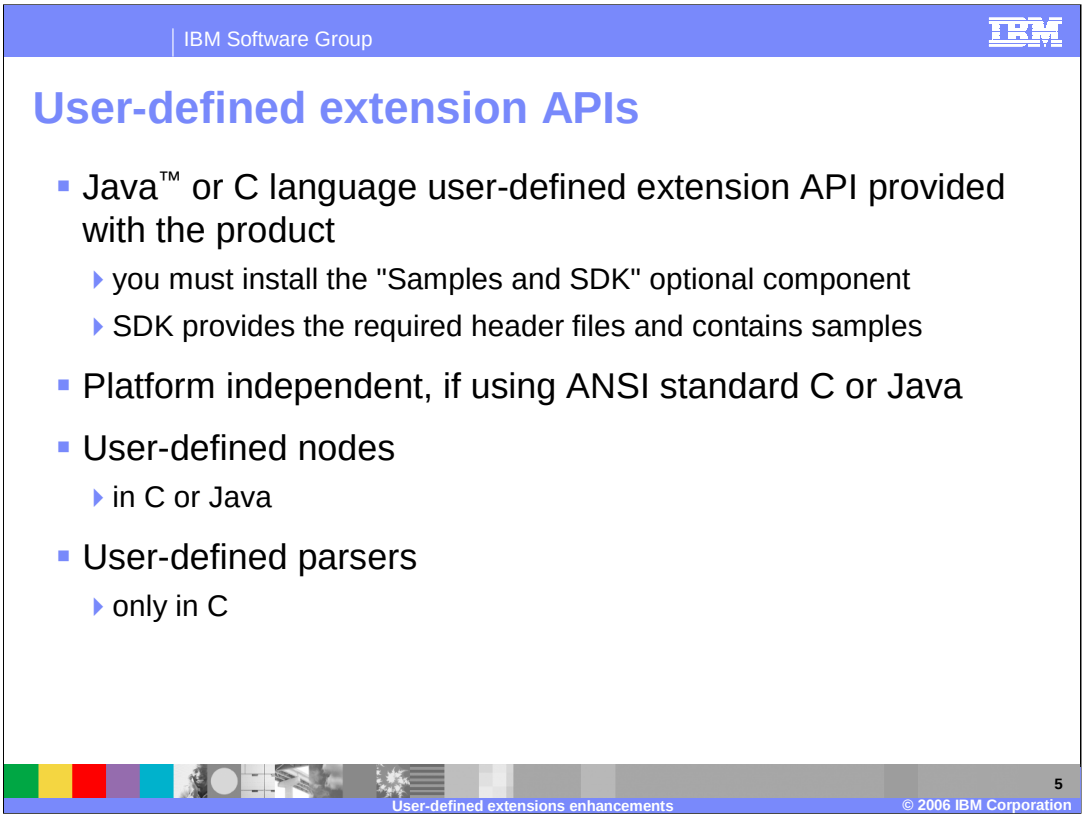
This section provides an overview of user-defined extensions.

User-defined extensions

- Add function to your implementation of WebSphere Message Broker
- Types of user-defined extensions:
  - User-defined input nodes
  - User-defined message processing nodes
  - User-defined output nodes
  - User-defined parsers

A user-defined extension is a component that you can design and implement to add to the function of your implementation of WebSphere Message Broker.

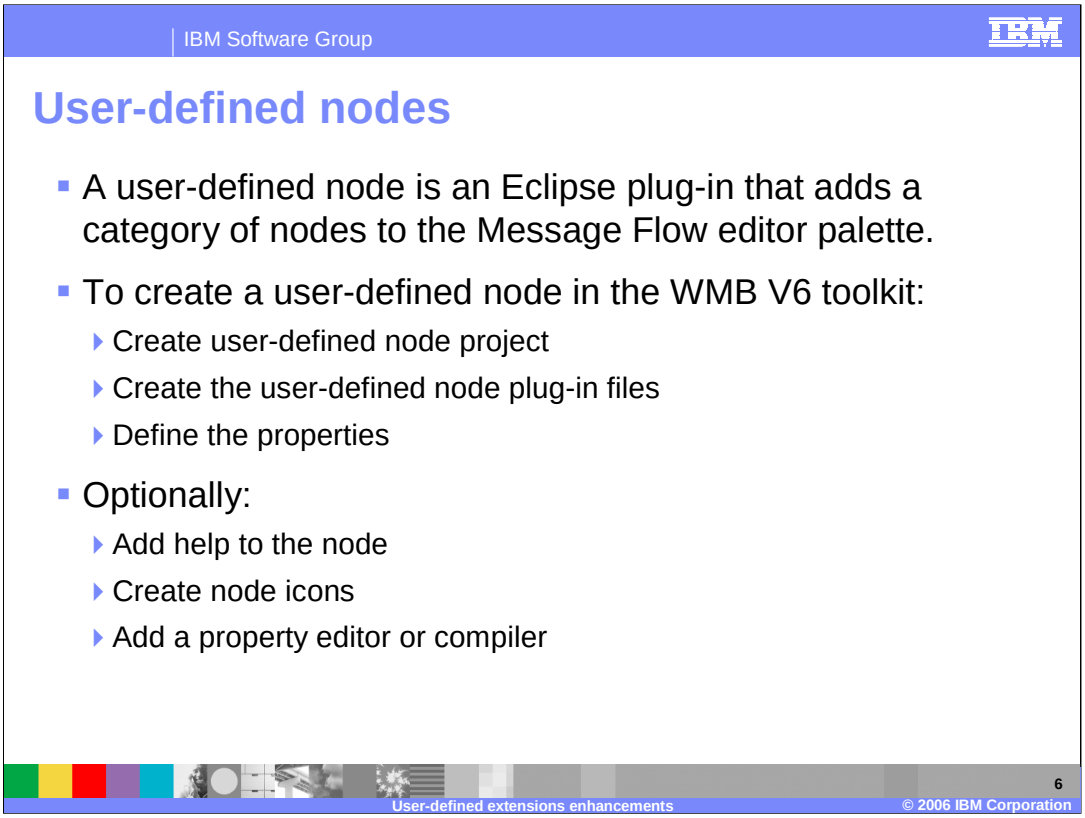
The user-defined nodes and parsers that you create can be used in conjunction with both the nodes and parsers supplied with the product, and with third-party supplied nodes and parsers. You can also configure a user-defined node to use a user-defined parser that you have written rather than one of the supplied parsers.



A user-defined parser must be written in the C programming language. User-defined nodes can be written in the C or Java programming language. User-defined nodes and parsers written in C must be compiled into a loadable implementation library, that is, a shared library on Linux® and UNIX®, or a Windows® DLL. User-defined nodes written in Java must be packaged as a jar file. You must integrate any user-defined extension you create into the WebSphere Message Broker tools before you can use it.

You can use your new node types on more than one operating system, if you make them platform independent. You can achieve this platform independence by using the ANSI standard C or Java programming languages, and by avoiding the use of platform specific code in your user-defined extension.

If you plan to program using the supplied Java or C language user-defined extension API, you must install the "Samples and SDK" optional component on at least one system. The SDK provides the required header files and contains samples that you can modify to your own requirements.



A user-defined node is an Eclipse plug-in that adds a category of nodes to the Message Flow editor palette. These user-defined nodes, often called plug-ins, allow you to define your own specific but reusable functionality in your message flows.

Before creating a user-defined node, you must have created a user-defined node project. To create the visual representation of your user-defined node in the workbench, perform the following tasks:

Create the user-defined node plug-in files:

In the Broker Application Development perspective, launch the Plug-in node project wizard: Click File > New > Project... then Message Flow Node Development > Message Flow Plug-in Node Project and specify a filename for the node

Define the properties

You can also perform the following optional tasks:

Add help to the node

Create node icons

Add a property editor or compiler

Section

User-defined extension enhancements

This section will cover the user-defined extension enhancements.

V6 enhancements to user-defined extensions

- User-defined extension limitations in the previous release included:
  - Validation of data type and object passed by API calls to and from user-defined extensions was not performed
  - Tracing on behalf of user-defined extensions was not available

In the previous release of Message Broker, the data that was passed to and from a user-defined extension was not validity checked. Any bad parameters or invalid data passed could cause a variety of problems. In addition, there was no tracing available for user-defined extensions.

User-defined extension API parameter checking

- User-defined extension API now checks parameters for correct type and valid object
  - Provides earlier detection of errors
  - Improves broker runtime robustness
  - Helps determine if problem is in user-defined node or broker
- New returns codes from most user-defined extension API calls
  - Indicates problem with parameter
    - Examples: cci_INVALID_ELEMENT_OBJECT, cci_INVALID_NODE_OJBECT

The user-defined extension interface API was enhanced in Version 6 to allow user-defined nodes to be more robust and to allow errors to be detected in the API layer. This extra checking also assists in problem debugging so that you can determine if the error is likely to be in the user-defined extension or in the broker itself. This is achieved by a series of API checks which validity check the parameter types and objects related to the user-defined extension API calls. New return codes are provided to report the various errors that can occur.

Writing entries to broker trace

- User-defined extensions can now write to broker trace
  - Improves problem diagnosis
  - Simplifies tracing
  - Consolidated with User and Service Trace (caller specifies message catalog to use)
    - cciUserTrace()
    - cciUserDebugTrace() – trace if user trace set to DEBUG
    - cciServiceTrace()
    - cciServiceDebugTrace() – trace if service trace set to DEBUG

```
2006-01-14 12:48:31.736389   3968  UserTrace SwitchMSG2: Switch
element received name=action value =wrongValue.  No user action
required.
```


The user-defined nodes can now optionally write user, user-debug, service, or service-debug trace entries to the broker trace file. This improves problem diagnosis and simplifies tracing, since the user and system trace entries are consolidated into a single file.

User-defined parser trace entry

- User debug must be set “on” for this trace
- BIP4146: a debug message that is traced when invoking a user-defined parser utility function, where the utility function alters the state of a syntax element. This includes all utility functions that start with cpiSetElement*, where * represents all nodes with that stem.

Trace entry BIP4146 is issued when a user-defined extension parser utility function alters the state of a syntax element. User debug tracing must be set “on” for this trace entry to be written.

User-defined extension API parameter error tracing

- BIP4147: an error message that is traced when a user-defined extension passes an invalid input object to a user-defined extension utility API function.
- BIP4148: an error message that is traced when a user-defined extension damages a broker's object. Possible message broker objects include syntax element, node, and parser.
- BIP4149: an error message that is traced when a user-defined extension passes an invalid input data pointer to a user-defined extension utility API function.
- BIP4150: an error message that is traced when a user-defined extension passes invalid input data to a user-defined extension utility API function.
  - Example: BIP4150 User defined extension input parameter failed debug validation check. Input parameter 2 passed into function ‘cciFoo’ does not have a valid value.
- Environment variable must be set for these trace entries to be written
  - MQSI_UDE_ERROR_CHECKING = on/off (“on” is default)

In Version 6, when a parameter error or object error is detected, a report of the error is now written to the broker trace log. Notice the new trace entries provided to report these errors, ranging from BIP4147 to BIP4150. These trace entries are controlled by MQSI_UDE_ERROR_CHECKING environmental variable, which is set “on” by default.

User-defined extension invocation/exit tracing

- User debug must be set “on” for these traces
- BIP2233 and BIP2234: a pair of messages traced before and after a user-defined extension implementation function is invoked. These messages report the input parameters and the returned value.
- BIP3904: a message traced before invoking the Java evaluate() method of a user-defined node.
- BIP3905: a message traced before invoking the C cniEvaluate() implementation function (iFpEvaluate member of CNI_VFT) of a user-defined node.

Additional tracing is added in Version 6 to record key processing points for user-defined extensions. BIP2233 is issued just before the invocation of a user-defined extension, and BIP2234 is issued after the return from the user-defined extension. BIP3904 is issued before the Java evaluate method is called within a user-defined extension. BIP3905 is issued just before the C language evaluate implementation function is called within a user-defined extension. User debug tracing must be set “on” for these trace entries to be written.

More user-defined extension trace entries

- User debug must be set “on” for these traces
- BIP4142: a debug message that is traced when invoking a user-defined node utility function, where the utility function alters the state of a syntax element. This includes all utility functions that start with cniSetElement*, where * represents all nodes with that stem.
- BIP4144 and BIP4145: a pair of messages traced by certain implementation functions that, when invoked by a user-defined extension, can modify the internal state of a message broker's object. Possible message broker objects include syntax element, node, and parser.

Trace entry BIP4142 is issued whenever a user-defined extension alters a syntax element. BIP4144 and BIP4145 trace entries record when the user-defined extension can modify the state of a message broker’s object. User debug tracing must be set “on” for these trace entries to be written.

User-defined extension trace multi-byte support

- User-defined extension trace now supports multi-byte characters
  - Wide forms of APIs
    - cciUserTraceW()
  - Wide exceptions
    - cciGetLastExceptionDataW
    - cciThrowExceptionW

The user-defined extension trace API now supports multi-byte characters, called the “wide” form of the API. Related “wide” exception responses are also issued. The “wide” form is indicated by the capital “W” suffix of the API call or exception response.

Call-back registration

- User-defined extension node “registers” for a certain event
- User-defined extension node informed when event occurs
  - “No messages to process”, “Instance pooled”, “Instance ended”
  - Optimize resource usage when flow is not processing messages
    - Close open files
    - Lazy commit
- Register user call-back function using new user-defined extension API
  - Allows call-back state to be maintained by threadContext

Version 6 allows a user-defined extension to register for a particular event in the system. Once registered, the user-defined extension is notified when that event occurs. You can use this facility, for example, to close any open files or to commit changes when there are no messages to process or when a message instance ends. The registration uses the new user-defined extension API, allowing the call-back state to be maintained by the thread context.

Example: Call-back registration

```
cciMessageContext* messageContext = cniGetMessageContext(NULL,message);
cciThreadContext* threadContext =
                    cniGetThreadContext(NULL,messageContext);
static MyContext myContext={1};

cciRegisterForThreadStateChange(
                    NULL,
                    threadContext,
                    & myContext,
                    switchThreadStateChange,
                    CCI_THREAD_STATE_IDLE |
                    CCI_THREAD_STATE_INSTANCE_END |
                    CCI_THREAD_STATE_TERMINATION);
```


Here is an example of a registration for call-back, which monitors for thread idle, thread instance end, and thread termination.

ESQL path expressions

- Allows user-defined node to build an ESQL path expression
  - User-defined node property specifications of runtime behavior
  - Navigation to input or output tree elements
- Examples
  - Navigate to **`InputRoot.XML.A.B[3].E`**
  - Navigate and create **`OutputRoot.XML.A.B[<].E.F`**

ESQL Path Expressions are allowed in WebSphere Message Broker Version 6. For example, you can create the path expression for a particular array element for input or output. Using path expressions allows the path to be more dynamic and can also reduce the number of function calls, especially if a target element is used a number of times and is deep within the tree structure.

ESQL path expression APIs

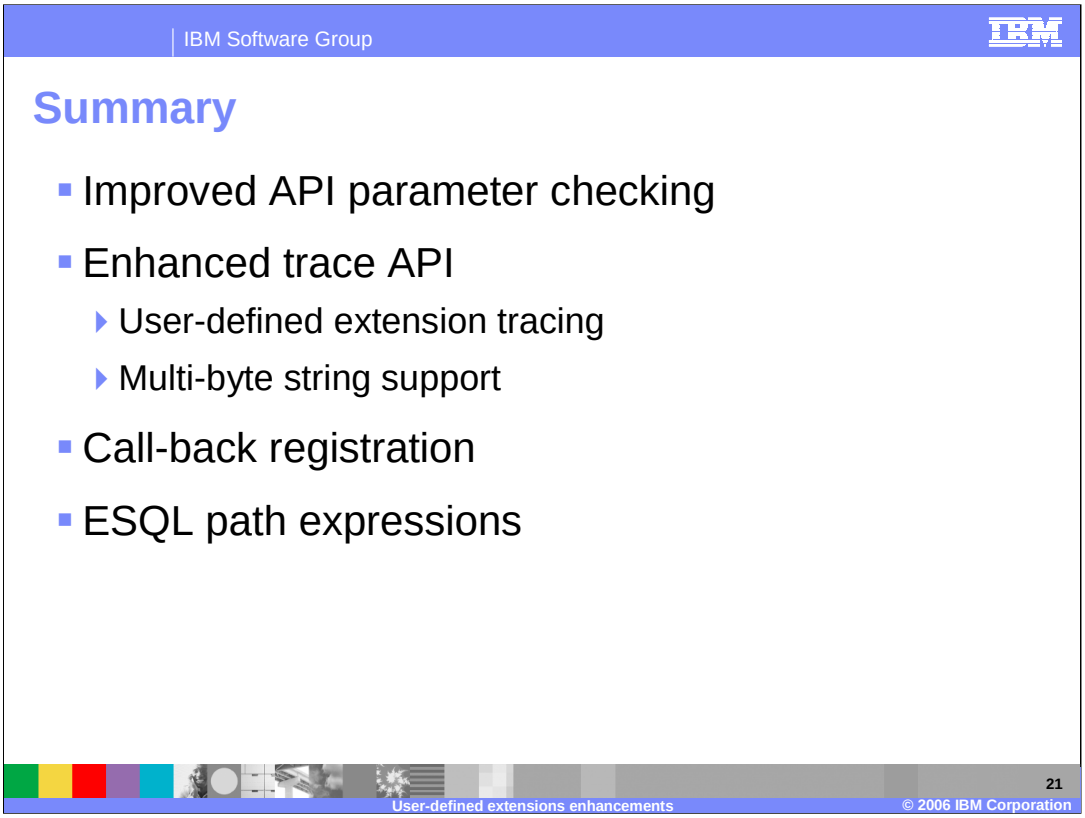
- Create path expression
  - cniSqlCreateReadOnlyPathExpression()
  - cniSqlCreateModifiablePathExpression()
- Run expression
  - cniSqlNavigatePath()
    - Returns located tree element
- Delete Expression
  - cniSqlDeletePathExpression()

There are two API calls to create the path expression objects; the CreateReadOnlyPathExpression and the CreateModifiablePathExpression. Once created, the NavigatePath API call is used to run the path expression that you have created. The DeletePathExpression API call is used to delete a path expression.

Section

Summary and references

The last portion of the presentation contains a summary and references.



WebSphere Message Broker provides enhancements in Version 6 for the user-defined extension API so that data references and objects are validity checked in the API interface. The enhanced API for user-defined extensions also provides additional tracing capabilities so that user-defined extension errors and optional user trace entries are written to the broker trace log. In addition, tracing and exception responses now support multi-byte strings.

A new call-back registration is provided so that user-defined extensions can monitor for certain events, allowing user-defined extensions to respond more dynamically to changes in the execution flow or system conditions.

ESQL path expressions are now supported which can reduce the number of API calls and allow more flexibility in path references.

References

- WebSphere Message Broker library is at:

http://www.ibm.com/software/integration/wbimessagebroker/library/

- WebSphere Message Broker Information Center is at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

- In the WebSphere Message Broker V6 Information Center:
  - See topic as04500 in the WMB V6 Information Center for details on creating a user-defined node
  - See topic as01380 for designing a user-defined extension
  - See topic au20120 for details resolving problems with user-defined extensions

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere z/OS zSeries

J2SE, Java, Javadoc, JDBC, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance 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 or performance improvements equivalent to the ratios stated here.