

OpenMedia™ is Commetrex’s implementation of the MSP Consortium’s M.100 recommendation. M.100 is a media-processing API that separates media-processing software from the underlying hardware, fostering independent competition and development in these two new computer-telephony value-adding layers.

M.100 allows vendor-independent development of M.100-compliant media-processing software and hardware resources. This means that with M.100 any "board" vendor can develop and market M.100-compliant hardware products and any media-processing technology vendor can develop and market compliant software products, substantially reducing the investment required to integrate media-processing software onto a common integrating hardware resource. What MVIP did at the PC level in 1990, M.100 does at the board or host level today.



OpenMedia is an open multi-stream, media-processing software environment that supports IP media-stream processing for voice-over-IP (VoIP) and fax-over-IP (FoIP), as well as traditional computer telephony circuit-switched applications.

OpenMedia reduces the cost of developing and porting media-processing products and supports straightforward media software deployment on Commetrex’s Media Stream Gateway boards, host PCs, and other hardware platforms. OpenMedia can be seamlessly integrated with the Open Telecommunications Framework (OTF) Kernel, Commetrex’s implementation of the ECTF S.100 host-level environment recommendation, as well as other vendors’ software environments.

OpenMedia can exist on any stream-processing computing resource or resources, making it just as applicable to proprietary embedded systems as to

general-purpose DSP-resource boards, or even a host PC. Commetrex is offering OpenMedia for license in three versions for

- Commetrex Media Stream Gateway products,
- Windows NT PC host implementations, and
- Proprietary-platform implementations.

Flexible Media Processing

OpenMedia is the only software environment that promotes the interoperability and portability of media-processing system resources provided by different vendors. It enables the developer to

- “Mix and Match” M.100-compliant media-processing resources from different vendors,
- Create proprietary integrated multiple-media products,
- Avoid years of software development, and
- Improve system performance by leveraging the industry’s most productive and resource-efficient multi-stream media-processing development environment.

Today, with M.100, the industry has the opportunity to integrate voice, fax, data, VoIP, FoIP, text-to-speech, speech recognition, and even video, on open low-cost media-processing resources.

OpenMedia Design Objectives

M.100 specifies an API; it only implies an implementation. The implementation goals of OpenMedia were

- System resource efficiency
- Development efficiency
- Skill-set separation
- Hardware independence and multi-processor support

In a signal-processing system resource efficiency means that DSP MIPS and on-chip RAM are conserved. The easiest way to improve MIPS efficiency is to limit the system to one media-processing task. Multiple media, with widely varying resource requirements, require OS-like task management on the DSP. But with media-diversity an overriding requirement for OpenMedia, an efficient resource manager to support multiple media is an absolute requirement.

Development efficiency is promoted when functions common to most applications are factored out of the application and placed in the supporting environment, an underlying principal of M.100. In OpenMedia this concept extends across all processors in the Environment: the host PC, the MSP Media Gateway's co-processor and its two DSPs.

Skill-set separation also promotes development efficiency. Most algorithm developers are not well-trained computer scientists. An effective environment will separate the DSP software from the rest of the system so the algorithm developers need not have system-level concerns.

Even with recent advances in DSP compiler design, most DSP algorithms are written in assembly code, which, by definition, is hardware-specific. But aside from this, there is no reason a multi-stream environment cannot be hardware independent. This means any stream-transform software in the system can run on any processor in the system transparent to the control program.

The MSP Environment

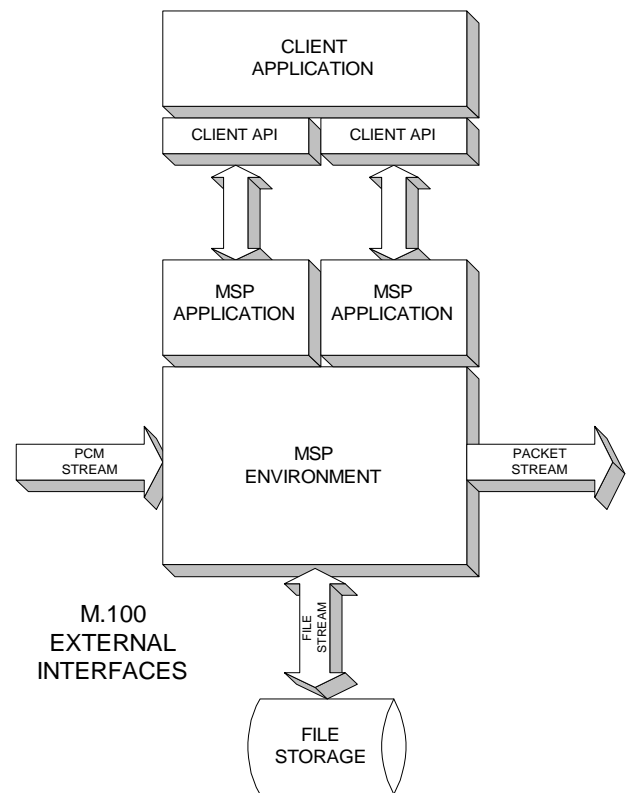
The software environment specified in M.100 is a logical aggregation of software components that control the flow of Media Streams and the operation of Media Stream Transforms (MSTs) on those streams. A Media Stream is a unidirectional flow of data between processing components with real-time constraints that usually involve the pacing requirements of isochronous ("equal-time") data. MSTs are software components that process Media Stream data in some way: change the format (transcoding), extract information (a modem), or add information to the Media Stream (modems, vocoders, PCM-to-linear conversion, automatic speech recognition).

The environment's components comprise a processing system that may include single or multiple processors: Pentium host processors, board-level co-processors, and DSPs are typical examples. One of the environment's

key software components is the MSP Application, responsible for requesting stream interconnections and bringing MSTs into execution to deliver a specified function to an external client entity.

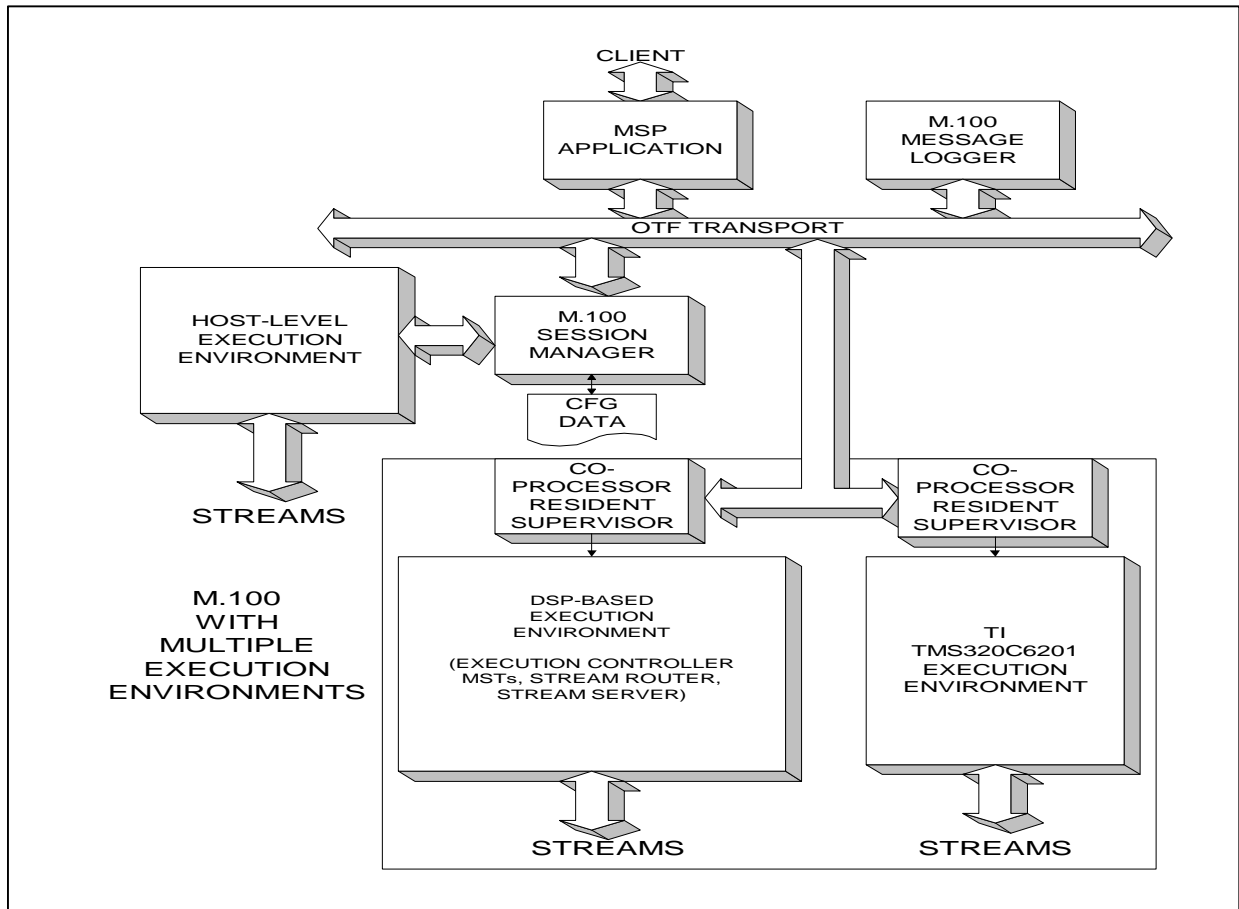
Streams enter and leave the MSP Environment through Stream Servers, which may be of various types: file stream servers that source or sink isochronous media (time-varying media in S.100 speak), image files for fax (spatial media), and text-based vocabularies for text-to-speech to and from persistent (file) storage. PCM Stream Servers interface PCM highways, such as H.100, to the MSP Environment. Packet Servers source and sink packets in IP applications.

An MSP Environment that serves as a gateway between a circuit-switched network and a packet network is shown in the figure below. Notice the only interfaces to "the outside world" are the three stream interfaces and the MSP Applications.



Stream-Paced Execution

An important M.100 concept is that of Stream-Paced Execution. "Pacing Streams" have real-time constraints. Resource efficiency can be improved if the natural processing rhythm of the MST is matched to that of the availability of Pacing Stream data. MSTs are required to



specify to a Packaging Utility an optimum amount of data to process. This information is then used by the environment (Execution Controller in OpenMedia) to start an MST's execution when an integral number of the specified buffers are available. Once the MST has processed the specified amount of data or an integral multiple of it, the MST relinquishes the processor without incurring preemption overhead.

MSP Applications are responsible for implementing and exposing the environment's functionality to client processes. In OpenMedia each MSP Application implements related functions, such as voice play/record or fax send/receive. Other components route streams, while others (the MSTs) implement stream-processing algorithms. In OpenMedia, these components can span multiple processors as shown above.

OpenMedia includes a Stream Router, which transports streams between board- and host-level execution environments. A fax send/receive facility could, therefore, be implemented with the fax modems (V.21, V.17, etc.) and image conversions on DSP-based execution environments and the fax protocol (T.30,

T.37, T.38, etc.) on the host processor. Low-port-count applications may be easily (and inexpensively) implemented on the host. At Commetrex, all MSTs are first coded and implemented in C, and validated in the host's execution environment, so host signal processing, which is extremely price-competitive in small systems, is a "free" fallout of this development approach.

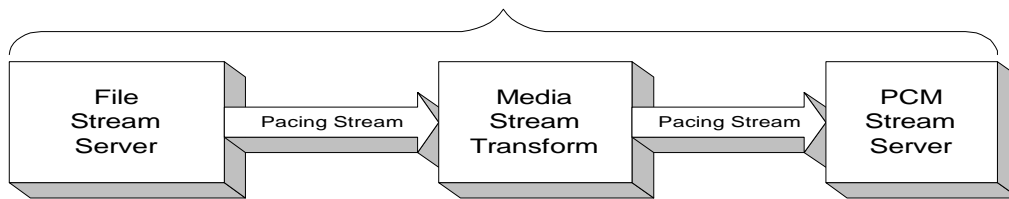
Packaging Utility

M.100 specifies an MSP Packaging Utility (MPU) that processes a set of directives to combine the executable elements of the MSP environment with descriptive parameters to produce files suitable for loading and execution. The MPU will typically define the entire environment, but functions are available to dynamically modify the Environment's resources.

Session Manager

The OpenMedia Session Manager (MSES) handles all service requests from an MSP Application. MSES is responsible for assigning these service requests to an Execution Environment. MSES tracks the resources available in each Execution Environment and assigns

Stream Processing Graph



them to “load-balance” the system. MSES learns of the resources it can control from the Packaging Utility or by registration commands from MSTs and Stream Servers.

All functions of an MSP process—MSP Applications, MSTs, and Stream Servers—are performed in the context of a Stream Session. An MSP Application can establish more than one session (define more than one Stream Graph). A Session context maintained by MSES describes the Graph with descriptions of system resources, Streams, and MSTs that are a part of that Processing Session. Once a Stream Session/Graph is defined, the MSP Application uses start/stop Graph commands to control the actual stream processing.

Stream Interface

OpenMedia is a stream-processing environment. All data, media, commands, and events are conveyed between OpenMedia entities via MSP Streams. A Stream is a unidirectional flow of data with one writer and one or more readers. All Streams have a fixed data type, and, if pacing streams, a fixed data rate. A Pacing Stream is a Stream that drives the rate of execution of an MST and the Stream Servers attached to it. A PCM stream is an example of a Pacing Stream. Ancillary Streams are Streams that are not Pacing Streams, such as command or event streams. Streams enter and exit the Environment through Stream Servers, which are environment and implementation specific.

The M.100 Interface has two components: The Stream API and the Stream Processing Interface. The Stream API provides functions to manage Stream sessions, describe streams, and write/read Ancillary Stream data, usually by MSP Applications. The Stream Processing Interface is specifically designed to implement an M.100 environment’s Pacing Stream functionality, and serves as the interface between the Execution Controller and Stream Graph elements that transfer Pacing Stream data. A Stream Process entry point is published by a Stream Server, Stream Router, or MST for each Pacing Stream interfaced. It is called repeatedly by the Execution

Controller to move buffers of Pacing Stream data between Stream Graph entities.

Stream Servers

OpenMedia supports several Stream Servers: File Stream Servers source and sink Streams from and to disk storage. PCM Stream Servers interface with H.100/110 PCM highways and the MSP-H8 line-interface card. Packet Stream Servers source and sink packet-based media data. Finally, Conference Bridge Stream Servers implement the special requirements of both PCM- and packet-based conference bridges.

Stream Router

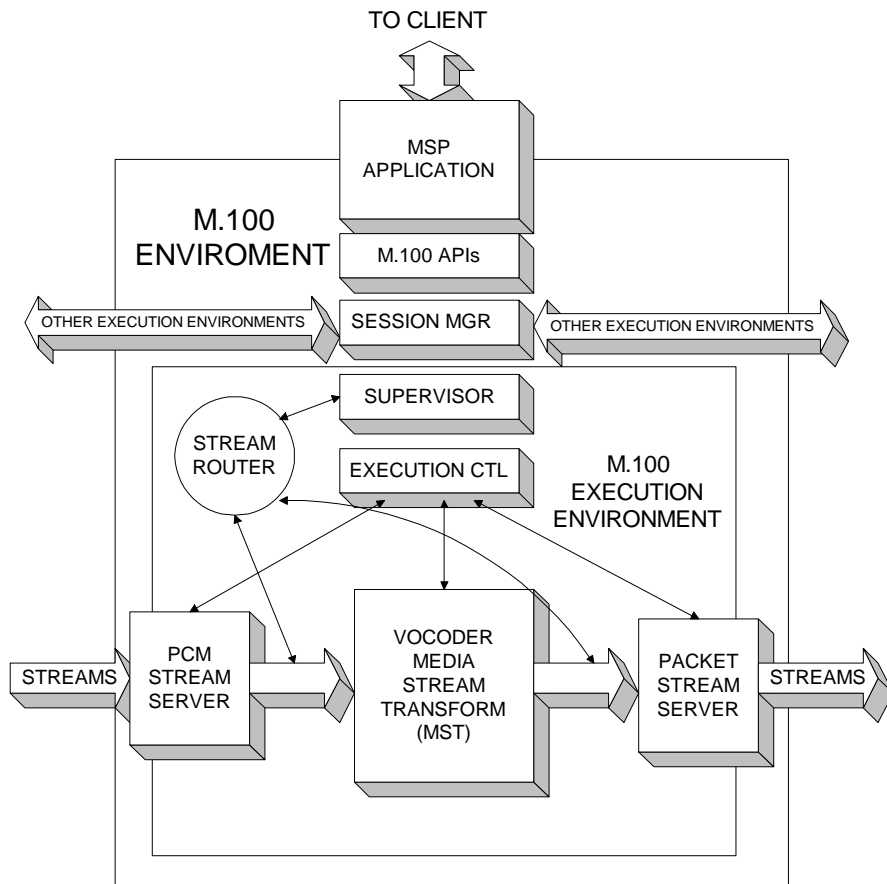
A Stream Router’s job is to isolate the environment’s inter-Execution Environment communication mechanism. In OpenMedia a Stream Router exists on the NT host and, if an MSP Media Gateway board exists in the system, on each Execution Environment on the board. Since more latency exists between the host and board-level environments an extra “slip-joint” is provided by the Stream Router in the form of flow-control tokens. This allows the Stream Router to fetch data ahead of the pacing requirements of the next element in the chain, usually an MST, up to the limits imposed by the board’s data buffering capabilities.

Execution Environment

The work of media processing is accomplished in an MSP Execution Environment, the dispatchable unit managed by the Session Manager (MSES). An OpenMedia Execution Environment is comprised of the resident MSP Supervisor (MSPV), an Execution Controller (MEC), a Steam Server and Router, and MSTs. MSES determines which Environment should execute a function and sends the command to that Environment’s MSPV. The MSPV manages the local resources, including MSTs, Stream Servers, and a Stream Router, if present. Each media processor on an MSP Media Gateway has an Execution Controller (MEC) that manages the execution of MSTs and Stream Servers that are located on the processor. The MEC also implements memory management and stream buffering.

On the MSP-320 the TMS320C6201 has a high-performance PCM serial port that interfaces directly to the board's H.100 PCM interface, requiring that a PCM Stream Server execute on the DSP.

Servers. It also implements memory management and stream buffering. The MEC is controlled by commands from the MSPV or from API calls made by the MSTs and the Stream Server and Router under its control.



Media Stream Transforms

The M.100 recommendation defines the MST: “A Media Stream Transform (MST) is a software function operating on a set of Media Streams and presenting a Stream Processing API and MST API to the MSP Environment.” During MSP Environment initialization, the available MSTs are defined to the MSP Environment by their Media Stream Transform Prototypes.

An MST Prototype is a software construct that identifies executable code and data, and provides a description of the processing requirements and Media Streams processed by the MST. This description is used by the MSP Environment to bring an MST into execution on a processor resource. The MSP Environment also uses the description to connect the MST to streams in the session and to control the execution of the MST.

The MST Prototype description includes the processor type, and the CPU cycle and memory requirements of the MST. It also defines all of the streams the transform takes as inputs and outputs. The MSP Environment uses the resource requirements and stream blocking information to schedule the execution of the MST.

Supervisor (MSPV)

Each of the two Supervisors (MSPV) located on the MSP-320's co-processor (and the MSPV on the NT host) manages a single Execution Environment. They are designed to offload processing from the DSP-based Execution Controller by managing the interface with MSES. By managing the interface and thus hiding the specifics of the Execution Environment MSES can operate on different environments transparently. Some commands result in an Execution Controller's dispatch queue being modified by the MSPV via the DSP's host-port interface.

Execution Controller

The M.100 Execution Controller (MEC) manages the execution of the MSTs, Stream Routers, and Stream

MST Portability and Interoperability

One of the key objectives of M.100 is media-processing software portability. The M.100 specification makes it possible for developers of stream-processing technology to independently develop and market M.100-conforming MSTs that will operate cooperatively on MSP Streams. This means software media-processing components can be even more portable than hardware components.

All communications between an MST and the Environment are through Streams. Since Streams are precisely defined in the M.100 specification, the Environment can “hook up” MSTs obtained from different vendors, and, as long as the output stream of one matches the Stream definition of the input of the

downstream MST, they will work cooperatively. This means MSTs can be developed to the M.100 standard, and then packaged for a specific environment by that environment's Packaging Utility.

Message Logger

The Message Logger accepts activity reports and error messages from the various components of the OpenMedia environment. The Message Logger is an independent task connected to the transport used by OpenMedia, usually the OTF Transport. MSES and MSPV open connections directly to the Message Logger. Messages, which are time stamped at the source, are posted by the Message Logger to a log file and optionally to a screen display. There can be one or more Message Loggers on a system, allowing all environments to log to the same file, or for each environment to have a separate log file.

Adding Media Resources with the OpenMedia SDK

The OpenMedia SDK allows you to easily add a media resource to your OpenMedia-based system by developing the MST, its MSP Application control program, and a client API. If a Stream Server is required that is not part of the supplied environment you will develop it using the Stream Server template supplied in the SDK. The SDK includes source-code shells for MSTs, MSP Applications, and Stream Servers, as well as working examples.

Adding a media resource to a OpenMedia system consists of the following steps:

1. Set system requirements
2. Define MST execution strategy
3. Set stream definitions
4. Complete packaging specification
5. Develop MST
6. Develop the MSP Application

Developer's Kit Choices

Commetrex offers three OpenMedia SDKs:

1. **MSP-320 OpenMedia** – This SDK supports development of media-processing software on the Commetrex Media Gateway MSP-320 DSP-resource board
2. **Portable OpenMedia** – This SDK provides the tools needed to implement OpenMedia on your proprietary board-level environment
3. **OpenMedia Host** – The SDK that delivers OpenMedia functionality to Window NT hosts without requiring an add-in processing board

MSP-320 OpenMedia SDK

The MSP-320 OpenMedia Developer's Kit gives the designer all the OpenMedia environment software tools needed to develop stream-processing DSP software for the MSP-320. The Kit includes the same software tools used by Commetrex's development engineers to develop Media Stream Transforms (MSTs) and MSP Applications.

Related Software Developer Kits

MSP-320 Open Telecommunications Framework
(OTF) Kernel – PN 20004

MSP-320 OTF Media Processing Software
MSP-320 PowerCall – PN 20005
MSP-320 PowerVOX – PN 20006
MSP-320 PowerFax – PN 20003

Commetrex, OpenMedia, Media Stream Gateway, Open Telecommunications Framework, OTF, and MSP-320 are trademarks of Commetrex Corporation. All other trademarks are the property of their respective holders. Specifications subject to change without notice.

Commetrex Corporation

6400 Atlantic Blvd.
Suite 190
Norcross, GA 30071
Voice: (770) 449-7775
Fax: (770) 242-7353
e-mail: marketing@commetrex.com
<http://www.commetrex.com>