

**Mint<sup>Multi-Tasking</sup>MT** Application Note**AN00102-002 - Embedded Programming Overview****Related Applications or Terminology**

- *Program the controller in 'C'*
- *Speedup execution time*
- *More memory for user code*
- *Customise the servo loop*
- *Generate new move types*

**Supported Controllers**

<b>NextMovePCI</b>	<input checked="" type="checkbox"/>
<b>NextMoveBX<sup>II</sup></b>	<input checked="" type="checkbox"/>
<b>NextMoveST</b>	<input checked="" type="checkbox"/>
<b>NextMoveES</b>	<input checked="" type="checkbox"/>
<b>NextMoveESB</b>	<input checked="" type="checkbox"/>
<b>MintDrive<sup>II</sup></b>	<input type="checkbox"/>
<b>Flex+Drive<sup>II</sup></b>	<input type="checkbox"/>

**Overview**

NextMove controllers are usually programmed using the MintMT programming language.

MintMT allows a user-friendly interface to the controller allowing the programming of simple motion control programs for commissioning machines right up to complex programs performing motion and IO synchronization for a complete machine.

For the control of a complete machine, there may be a small number of applications where MintMT does not provide sufficiently high execution speed or where some customization is necessary. In these situations an embedded application is ideal.

**What is an Embedded Application?**

Distributed with all NextMove controllers are the Embedded Developer Libraries. These are a collection of run-time libraries for the NextMove controller called the Mint Motion Library (MML). These libraries support all the function calls made by MintMT and can be used along with the Texas TMS320C31 cross compiler to create embedded applications with all the functionality of a MintMT program.

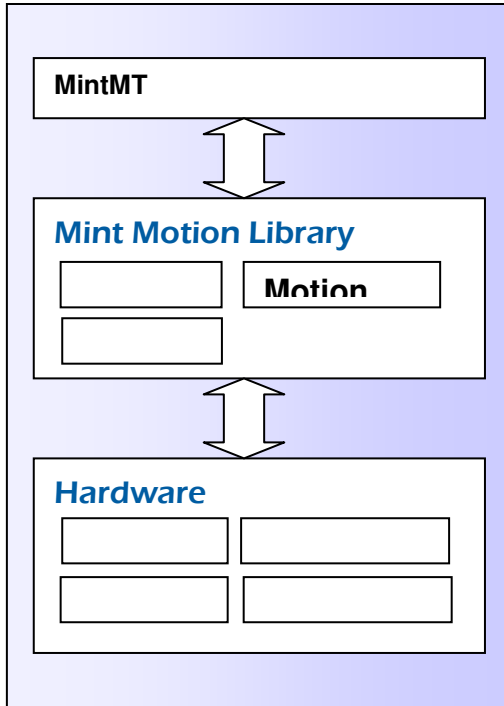
All embedded applications must be written in the 'C' programming language and compiled using the TI<sup>1</sup> compiler. Examples of embedded applications are provided with the Embedded Developer Libraries. These contain build files and the memory map for the controller and can be used as a template for developing an embedded application.

An embedded application replaces MintMT on the controller, however the ICM communication layer is still supported, this means that both the Mint Integrated Development Environment (WorkBench v5) and function calls via the ActiveX control will still function. Workbench v5 can therefore be used as a debugging tool when developing the application and a PC-based operator interface can be implemented if required.

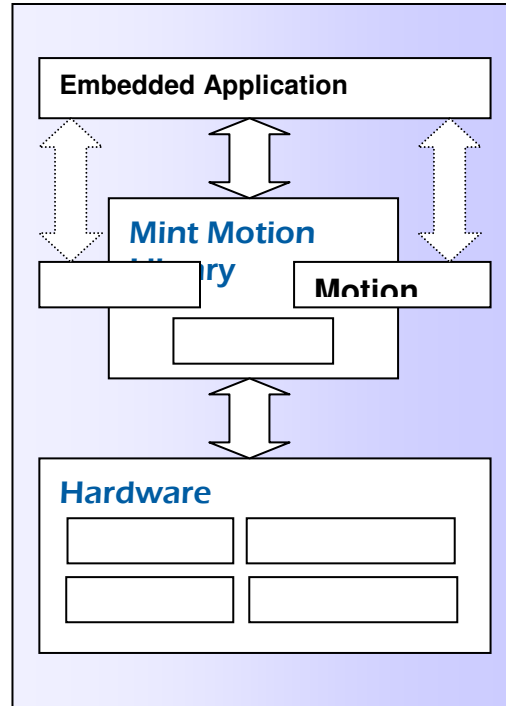
<sup>1</sup> Texas Instruments

### NextMove Architecture

*Controller running MintMT*



*Controller running an Embedded Application*



As MintMT is not required when writing an embedded application the available memory for embedded applications is greater than that available by MintMT programs.

# Mint<sup>Multi-Tasking</sup> MT Application Note

## *Converting from Mint MT to Embedded*

The API calling convention of all MML function calls remains consistent within MintMT. This allows MintMT programs to be converted to 'C' relatively easily.

**For example:**

MintMT Code	'C' Code
<code>NODETYPE(2,14) = _ntKEYPAD</code> <code>PAUSE NODELIVE(2,14)</code>	<code>SetNodeType (2, 14, ntKEYPAD) ;</code> <code>Do</code> <code>{</code> <code>    live=GetNodeLive (14) ;</code> <code>}</code> <code>While (!live);</code>
<code>MASTERSOURCE(0) = _msAUXENCODER</code> <code>FOLLOWMODE(0) = _fmNO_RAMP</code> <code>FOLLOW(0) = 2</code>	<code>SetMasterSource (0, msAUXENCODER) ;</code> <code>SetFollowMode (0, fmNO_RAMP) ;</code> <code>SetFollow (0, 2.00) ;</code>
<code>DAC(2) = 0</code> <code>OUTX(0) = 1</code> <code>ADCMODE(4) = _acBIPOLAR</code>	<code>SetDAC (2, 0) ;</code> <code>SetOutX (0, 1) ;</code> <code>SetADCMode (4, acBIPOLAR) ;</code>

This means that all axes can be initially configured and the application prototyped using Mint and WorkBench v5. This startup code can then be converted from Mint to 'C' for use in an embedded application. As there is no change in functionality due to the common API there is a high confidence of error free code.

## *Execution Speed Up*

As embedded applications are written in C rather than Mint they are not subject to the overhead of the MintMT virtual machine. All non-MML calls are compiled, by the native C31 compiler, down to several instructions.

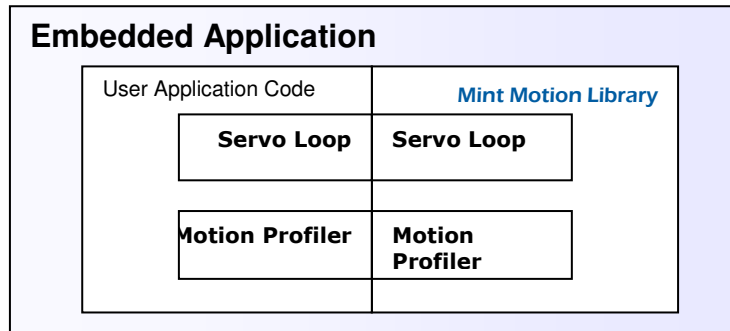
Within MintMT, performing an arithmetic operation on two variables may take 10's of microseconds. Within an embedded application this is executed in several instruction cycles giving a speed improvement of about 100-200 times.

The MML functions still have to execute the same embedded code, but the execution time for these is still improved due to the removal of the MintMT virtual machine. MML functions will generally execute about 2-5 times faster than MintMT when using an embedded application.

# Mint<sup>Multi-Tasking</sup> MT Application Note

## Open Architecture

As embedded applications are written in 'C', this allows certain sections of the MML core to be overwritten or customized. The open architecture allows the servo loop and motion profiler for any/all axes to be modified. The use of a customised servo loop or motion profiler does not prevent the use of the default servo loop or motion profiler.



## Servo Loop

The servo loop is the position control loop that runs on all servo axes when they are enabled. It reads the current position of the axis, compares it to a demand position and from the resulting difference (following error) uses a control law to generate a demand (DAC) output to move the axis closer to the demand position. Open Architecture allows a new servo loop to be called in place of the Baldor servo loop if required.

By default the servo loop is executed every 1ms, although for tighter control it may be necessary to reduce the time between servo loop execution.

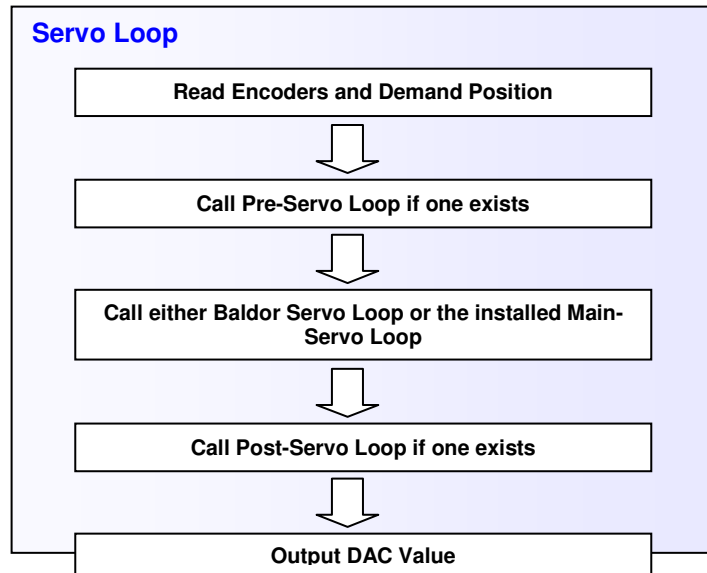
There are three installable servo loop options, these are:

- Pre Servo Loop** This is called directly after the actual and demand positions have been calculated, but prior to the point that the following error is calculated. This is designed to allow users to overload either the actual position (read from the encoder) or the demand position (generated by the motion profiler).
- Main Servo Loop** This is called in place of the PIDVFA control law that is used to generate the DAC demand value. This is designed to allow users to write their own control law.
- Post Servo Loop** This is called directly before the DAC value is written to the hardware. This is designed to allow a DAC filter to be implemented.



# Mint<sup>MT</sup> Multi-Tasking Application Note

The following diagram represents the Servo Loop stages.



Customized servo loops are installed for specific axes. This means that some of the axes can use the default Baldor servo loop while others have a customized servo loop. Some may have a post-servo loop and others a pre-servo loop.

## Motion Profiler

The motion profiler runs for each axis in turn and checks if a move type is loaded and triggered. If so it will call the specific trajectory generation routine for that move type. Open Architecture allows new trajectory generation routines to be called by the motion profiler. The trajectory generation routine generates a new demand position for one or more axes, this new position is then used by the servo loop / stepper loop to generate an appropriate control output.

By default the motion profiler is executed every 2ms, although for tighter control it may be necessary to reduce the time between execution.

Installing a new trajectory generator does not replace the existing routines such as homing or linear moves but is an addition to them. A separate trajectory generator may be installed for each axis or multiple axes may use the same one. Alternatively a single trajectory generator may generate a demand position for multiple axes.

The new move type must be triggered using the **GO** command, this allows synchronization with other axes performing motion.