



ARCHIVED SITE (NOT VISIBLE TO PUBLIC)

Search

Embedded Beat » (<http://blogs.freescale.com/>)

Internet of Things(<http://blogs.freescale.com/category/iot/>)

Automotive(<http://blogs.freescale.com/category/automotive/>)

Networking(<http://blogs.freescale.com/category/networking/>)

Microcontrollers(<http://blogs.freescale.com/category/mcus/>)

Processors(<http://blogs.freescale.com/category/processors/>)

Sensors(<http://blogs.freescale.com/category/sensors/>)

Freescale.com(<http://www.freescale.com>)

January 23, 2013(<http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/>)

Latest Posts

3 reasons why Kinetis MCUs offer a smarter approach to security

Orientation Representations: Part 2

In Orientation Representations: Part 1(<http://blogs.freescale.com/iot/2012/10/orientation-representations-part-1/>), we explored the use of rotation matrices and Euler angles. At the end of that discussion, I alluded to the fact that there might be more efficient ways of describing rotations. Let's start with the rotation of a simple rigid body (in this case a cylinder) as shown in Figure 1. Here, the cylinder is rotated such that a point on its surface originally at "A" is rotated to point "B" in space.

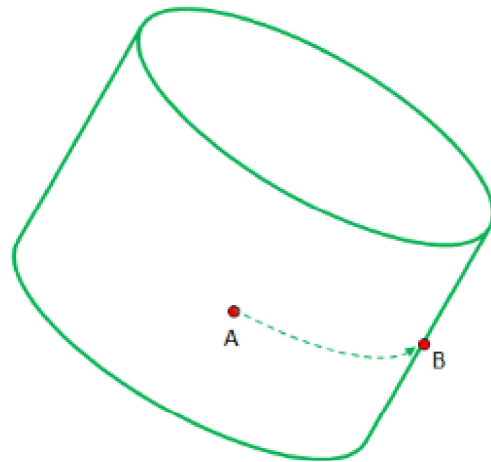


Figure 1: Rotation of a rigid body such that a reference point moves from "A" to "B"

For this simple case, I've kept the axis of rotation along the vertical axis of the cylinder as shown in Figure 2. But that is not a requirement for the underlying mathematics to work. So long as we have a rigid body, we can always describe the rotation in the manner that follows.

(<http://blogs.freescale.com/mcus/2015/11/3-reasons-why-kinetis-mcus-offer-a-smarter-approach-to-security/>)

Groove is in the wafer: The first-ever silicon record debuts
(<http://blogs.freescale.com/the-embedded-beat/2015/11/groove-is-in-the-wafer-the-first-ever-silicon-record-debuts/>)

Your next harsh environment application deserves a more robust solution
(<http://blogs.freescale.com/mcus/2015/11/your-next-harsh-environment-application-deserves-a-more-robust-solution/>)

Get started with ARM Cortex-MF4 based FRDM-K22F dev board that's Arduino R3-pin compatible
(<http://blogs.freescale.com/mcus/2015/11/get-started-with-arm-cortex-mf4-based-frdm-k22f-dev-board-thats-arduino-r3-pin-compatible/>)

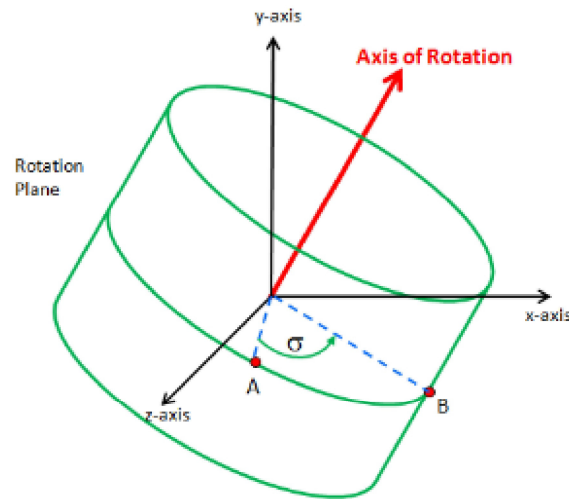


Figure 2: Overlay of Cartesian Coordinates onto System of Figure 1

Figure 3 deals with the same rotation, but focuses on the fact that we have a rotation plane that is perpendicular with the axis of rotation. The movement of the cylinder is a rotation equal of angle α , about the axis of rotation, where the point of interest is constrained to lie within the rotation plane.

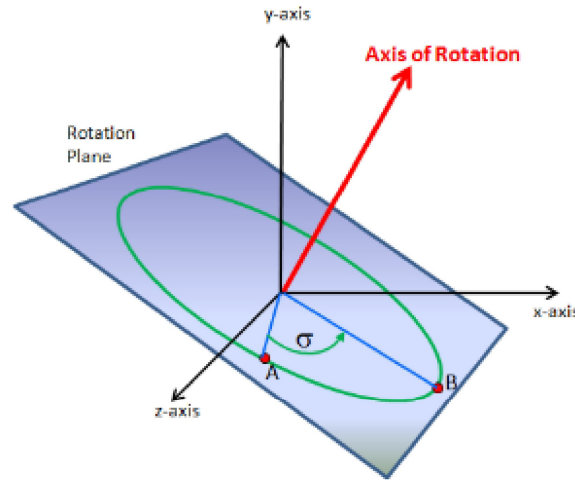
Auto IoT: Beyond the car and into the infrastructure (Part 3)
(<http://blogs.freescale.com/automotive/2015/11/auto-iot-beyond-the-car-and-into-the-infrastructure-part-3/>)

Breaking through the Wearable Clutter: Design News
(<http://blogs.freescale.com/iot/2015/11/breaking-through-the-wearable-clutter-design-news/>)

New York Times and Google turn cardboard into a new reality
(<http://blogs.freescale.com/sensors/2015/11/latest-sensor-fusion-application/>)

What's next for your wearables design? WaRP 7
(<http://blogs.freescale.com/iot/2015/11/whats-next-for-your-wearables-design-warp-7/>)

Focus on IoT with ARM mbed and Freescale Freedom platforms
(<http://blogs.freescale.com/mcus/2015/11/focus-on-iot-with-arm-mbed-and-freescale-freedom-platforms/>)



What you can do with Sensor Data Analytics, from babies to horses
(<http://blogs.freescale.com/sensors/2015/11/babies-to-horses-what-you-can-do-with-sensor-data-analytics/>)

Figure 3: Looking at Just the Rotation Plane and Axis of Rotation

The rotation is fully described by the three components of the normalized rotation axis and the rotation angle α , which may be in radians or degrees, depending upon the system in use. As an example, I recently did some OpenGL ES graphics programming. This system is very popular on portable devices. I'm using it to program demos for Android, for later porting to iOS. In OpenGL ES, you build up 3 dimensional objects as a collection of triangles, which can then be offset and/or rotated to change perspective. As an example, our cylinder might be crudely drawn as shown in Figure 4.

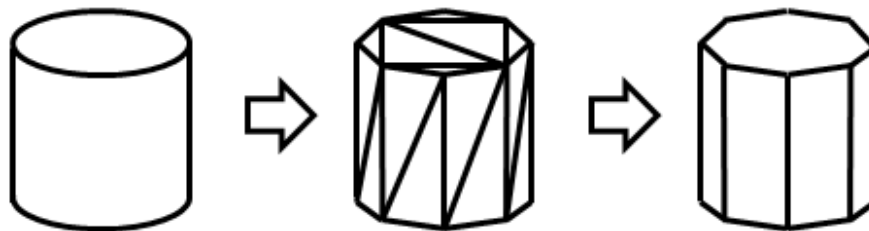


Figure 4: OpenGL ES Drawing of a Cylinder

In this case, I've modeled the top and bottom of the cylinder with 6 triangles each, and the other side is modeled using a total of 16 triangles arranged in a strip. OpenGL ES is optimized to draw such structures efficiently, and it is possible to then "render" textures onto the drawn surfaces. What's really neat is that once drawn, we get a reasonable approximation of the cylinder of Figure 1 simply by doing a -30 degrees rotation about the Z axis (presumed to be out of the page) using a single OpenGL ES instruction:

```
gl.glRotatef(-30.0f, 0.0f, 0.0f, 1.0f);
```

At this point, you're probably thinking: "Yeah, that makes sense, but how does it work at the math level?" This is the where I need to introduce the concept of a **quaternion**. Conceptually, a quaternion encodes the same axis and angle as above. But for mathematical reasons it deals with 1/2 of the rotation angle as shown below.

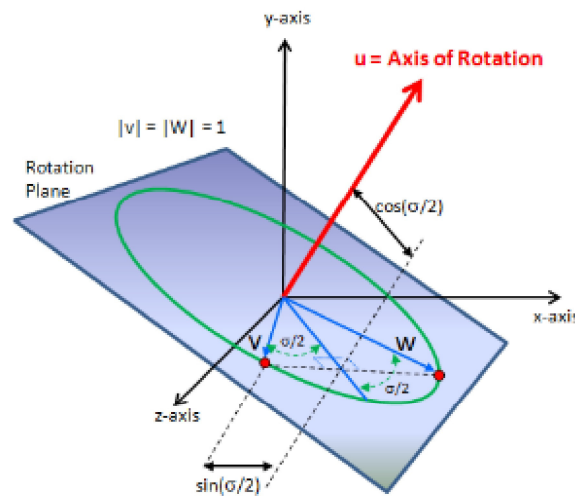


Figure 5: System of Figure 4 in Terms of Quaternion Components

Before overwhelming you with the underlying math, you should know that unless you are planning to implement your own quaternion utility library, you only need to know a few key points:

- 1) It takes four numbers to fully describe a quaternion (commonly q_0 through q_3).
- 2) Not all quaternions are rotation quaternions. Rotation quaternions have unit length ($q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$). The discussion below will be restricted to rotation quaternions.
- 3) These same rotations can be described using Euler angles, rotation matrices, etc. as discussed in the previous posting. It is possible (and common) to translate between formats and use multiple formats. Rotation matrices have the advantage of always being unique. Euler angles are subject to gimbal lock, and should not be used for internal calculations (only input/output of results).
- 4) You can rotate a vector \mathbf{V} using a quaternion q using the equation: $\mathbf{W} = q\mathbf{V}q^*$ (quaternion products and complex conjugates are defined later)
- 5) A sequence of rotations represented by quaternions q_1 followed by q_2 can be collapsed into a single rotation simply by computing the quaternion product $q = q_2q_1$ and then applying the rotation operator as above.

I will be presenting the mathematical definition first, and without proof. If you really, REALLY want to know the underlying theory, let me suggest that you pick up a copy of Jack Kuiper's excellent text: *Quaternions and Rotation Sequences*. This appears to be (by far) the most extensive treatment on the topic, even while remaining very readable.

Notice that rotation quaternions deal with $\alpha/2$, not α . We can define a rotation quaternion

“q” in one of several equivalent fashions.

$$\mathbf{q} = (q_0, q_1, q_2, q_3) \quad (\text{Eqn. 1})$$

$$\mathbf{q} = q_0 + \mathbf{q}, \text{ where } \mathbf{q} = iq_1 + jq_2 + kq_3 \quad (\text{Eqn. 2})$$

$$\mathbf{q} = \cos(\alpha/2) + \mathbf{u} \sin(\alpha/2), \text{ where } \mathbf{u} \text{ is the vector axis of rotation} \quad (\text{Eqn. 3})$$

I use the quaternion form where $q_0 = \cos(\alpha/2)$. Some texts will reorder the quaternion components so that the vector portion \mathbf{q} is contained in q_0 -2 and $q_3 = \cos(\alpha/2)$. Be sure you understand which form your text/software library supports.

Quaternions are a form of hyper-complex number where instead of a single real and single imaginary component, we have one real and THREE imaginary components (i, j & k).

Rules for these imaginary components are:

$$i^2 = j^2 = k^2 = ijk = -1 \quad (\text{Eqn. 4})$$

$$ij = k = -ji \quad (\text{Eqn. 5})$$

$$jk = i = -kj \quad (\text{Eqn. 6})$$

$$ki = j = -ik \quad (\text{Eqn. 7})$$

Two quaternions, p and q, are equal to one another only if the individual components are equal. You add two quaternions by adding the individual components. If

$$\mathbf{p} = p_0 + ip_1 + jp_2 + kp_3; \text{ and} \quad (\text{Eqn. 8})$$

$$\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3 \quad (\text{Eqn. 9})$$

Then

$$\mathbf{p} + \mathbf{q} = (p_0 + q_0) + i(p_1 + q_1) + j(p_2 + q_2) + k(p_3 + q_3) \quad (\text{Eqn. 10})$$

The addition operation commutes. That is $p+q = q+p$. Multiplication of a quaternion by a scalar real number is trivial, just multiply each of the four components by the scalar. Multiplication of two quaternions is NOT so trivial:

$$pq = p_0q_0 - \mathbf{p} \cdot \mathbf{q} + p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q} \quad (\text{Eqn. 11})$$

Multiplying one quaternion by another quaternion results in a third quaternion. Notice that the 1st two components ($p_0q_0 - \mathbf{p} \cdot \mathbf{q}$) makes up the scalar portion of the result, and the last three ($p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q}$) comprise the vector portion. The quaternion product operation is not commutative $pq \neq qp$. Order matters. Multiplication of two quaternions includes scalar, cross product and dot product terms. Unless you are writing your own quaternion library, you are likely never to use the expression above. Instead, you will use a function that does the quaternion multiplication for you.

The complex conjugate of

$$q = q_0 + iq_1 + jq_2 + kq_3 \text{ is } q^* = q_0 - iq_1 - jq_2 - kq_3 \quad (\text{Eqn. 12})$$

Related to this, we have

$$(pq)^* = q^*p^* \quad (\text{Eqn. 13})$$

$$q+q^* = 2q_0 \quad (\text{Eqn. 14})$$

$$q^{-1} = q^* \text{ for any unit quaternion} \quad (\text{Eqn. 15})$$

Eqn. 15 is interesting. If you think of a quaternion as a rotation operator, it says you can reverse the sense of rotation by inverting the axis of rotation. Given our usual standard of using the Right Hand Rule to describe the polarity of rotations, this makes perfect sense. Reversing the direction of the axis is equivalent to reversing the direction of rotation.

Another interesting take on the above is that rotation quaternions are not unique:

$$q = -q \quad (\text{Eqn. 16})$$

Any rotation quaternion can be multiplied by -1 and still result in the same rotation! That's because we reversed both the angle AND the axis of rotation (which then cancel each other). It is conventional therefore to remove the ambiguity by negating a rotation quaternion if its scalar component is negative.

At this point, you are surely wondering why in the world you might, or might not, choose to use quaternions instead of rotation matrices. Here's a brief summary of the pros and cons:

Topic	Quaternion	Rotation Matrix
Storage	Requires 16 bytes of storage in single precision floating point (4 elements at 4 bytes each)	Requires 36 bytes of storage (9 elements at 4 bytes each)
Computation (for 2 sequential rotations)	4 elements each requiring 4 multiplies and 3 additions = 28 operations	9 elements, each requiring 3 multiplies and 2 additions = 45 operations
Vector rotation	Rotating a vector by pre- and post-multiplication of quaternion requires 52 operations	Rotating a vector via rotation matrix requires 15 operations (3 elements each requiring 3 multiplies and 2 additions)
Discontinuities	Generally, we force the scalar part of the quaternion to be positive, which can cause a discontinuity in the rotation axis (it flips).	None
Ease of Understanding	Generally takes a lot of study to understand the details	Easily understood by most engineers

Topic	Quaternion	Rotation Matrix																		
Conversion	<p>From rotation matrix =</p> <table border="1"> <tr> <td>m11</td><td>m12</td><td>m13</td></tr> <tr> <td>m21</td><td>m22</td><td>m23</td></tr> <tr> <td>m31</td><td>m32</td><td>m33</td></tr> </table> <p>we have:</p> $q_0 = 0.5 \sqrt{m_{11} + m_{22} + m_{33} + 1}$ $q_1 = (m_{32} - m_{23}) / (4q_0)$ $q_2 = (m_{13} - m_{31}) / (4q_0)$ $q_3 = (m_{21} - m_{12}) / (4q_0) \quad (\text{Eqn. 17})$	m11	m12	m13	m21	m22	m23	m31	m32	m33	<p>RM =</p> <table border="1"> <tr> <td>$2q_0^2 - 1 + 2q_1^2$</td><td>$2q_1q_2 - 2q_0q_3$</td><td>$2q_1q_3 + 2q_0q_2$</td></tr> <tr> <td>$2q_1q_2 + 2q_0q_3$</td><td>$2q_0^2 - 1 + 2q_2^2$</td><td>$2q_2q_3 - 2q_0q_1$</td></tr> <tr> <td>$2q_1q_3 - 2q_0q_2$</td><td>$2q_2q_3 + 2q_0q_1$</td><td>$2q_0^2 - 1 + 2q_3^2$</td></tr> </table> <p>(Eqn. 18)</p>	$2q_0^2 - 1 + 2q_1^2$	$2q_1q_2 - 2q_0q_3$	$2q_1q_3 + 2q_0q_2$	$2q_1q_2 + 2q_0q_3$	$2q_0^2 - 1 + 2q_2^2$	$2q_2q_3 - 2q_0q_1$	$2q_1q_3 - 2q_0q_2$	$2q_2q_3 + 2q_0q_1$	$2q_0^2 - 1 + 2q_3^2$
m11	m12	m13																		
m21	m22	m23																		
m31	m32	m33																		
$2q_0^2 - 1 + 2q_1^2$	$2q_1q_2 - 2q_0q_3$	$2q_1q_3 + 2q_0q_2$																		
$2q_1q_2 + 2q_0q_3$	$2q_0^2 - 1 + 2q_2^2$	$2q_2q_3 - 2q_0q_1$																		
$2q_1q_3 - 2q_0q_2$	$2q_2q_3 + 2q_0q_1$	$2q_0^2 - 1 + 2q_3^2$																		

Equations 17 and 18 are consistent with regards to direction of rotation. If instead of rotating a vector in a fixed frame of reference, you rotate the frame of reference itself, you will need to use the transpose of Eqn. 18 and invert q_1 , q_2 and q_3 in Eqn. 17.

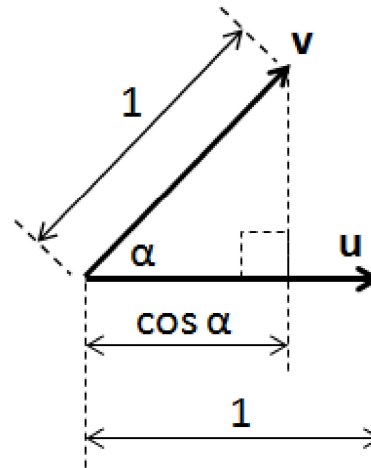
Returning to the quaternion rotation operator $\mathbf{W} = \mathbf{q}\mathbf{V}\mathbf{q}^*$, note that \mathbf{V} needs to be expressed as a quaternion of the form $[0, v_x, v_y, v_z]$, and the multiplications are quaternion multiplies as defined in Eqn. 11. q^* is the complex conjugate defined in Eqn. 12.

If you do a lot of graphics or sensor fusion work, you will probably find yourself constantly switching between the various representations we've considered. You'll find it useful to remember a couple of identities from your high school geometry course:

The Dot Product $\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}| |\mathbf{v}| \cos \alpha$ (Eqn. 19)

If both \mathbf{u} and \mathbf{v} are unit vectors, then:

$$\mathbf{u} \cdot \mathbf{v} = \cos \alpha \text{ (Eqn. 20)}$$

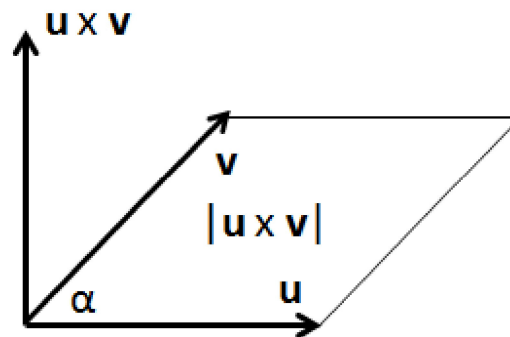


The Cross Product $\mathbf{u} \times \mathbf{v} = |\mathbf{u}| |\mathbf{v}| \sin \alpha \mathbf{n}$ (Eqn. 21)

where \mathbf{n} is a unit vector perpendicular to the plane containing \mathbf{u} and \mathbf{v} (the polarity of \mathbf{n} follows the right hand rule).

If both \mathbf{u} and \mathbf{v} are unit vectors, then:

$$\mathbf{n} = \mathbf{u} \times \mathbf{v} / (\sin \alpha) \text{ (Eqn. 22)}$$



If you've been paying attention, you will see that α is the rotation of \mathbf{u} into \mathbf{v} about the axis


of rotation defined by $\mathbf{u} \times \mathbf{v}$. See! It's simple! Axis and angle!

References:

- 1) Quaternions and Rotation Sequences, Jack B. Kuipers, Princeton University Press, 1999
- 2) Euler Angles(<http://demonstrations.wolfram.com/EulerAngles/>) from the Wolfram Demonstrations Project(<http://demonstrations.wolfram.com/>) by Frederick W. Strauch
- 3) Diversified Redundancy in the Measurement of Euler Angles Using Accelerometers and Magnetometers, Chirag Jagadish and Bor-Chin Chang, Proceedings of the 46th IEEE Conference on Decision and Control, Dec. 2007
- 4) "Euler Angles" at Wikipedia(http://en.wikipedia.org/wiki/Euler_angles)
- 5) Orientation Representations: Part 1 (<http://blogs.freescale.com/iot/2012/10/orientation-representations-part-1/>), Michael Stanley at the Embedded Beat, October 2012

Share this:  ([http://www.facebook.com/sharer.php?](http://www.facebook.com/sharer.php?u=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/)

[u=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/))

 ([https://plus.google.com/share?](https://plus.google.com/share?url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/)

[url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/))

 ([http://www.tumblr.com/share/link?](http://www.tumblr.com/share/link?url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/)

[url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/))



(<http://twitter.com/share?>

[url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/&text=Orientation+Representations%3A+Part+2+)

[2/&text=Orientation+Representations%3A+Part+2+](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/&text=Orientation+Representations%3A+Part+2+)



(<http://reddit.com/submit?>

[url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/&title=Orientation+Representations%3A+Part+2+)

[2/&title=Orientation Representations: Part 2\)](http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/&title=Orientation+Representations%3A+Part+2+)



(<http://www.linkedin.com/shareArticle?>

[mini=true&url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/](http://www.linkedin.com/shareArticle?mini=true&url=http://blogs.freescale.com/sensors/2013/01/orientation-representations-part-2/))

Sensors(<http://blogs.freescale.com/category/sensors/>), The Embedded Beat
(<http://blogs.freescale.com/category/the-embedded-beat/>)

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website	<input type="text"/>
Comment	<input type="text"/>
<input type="button" value="Post Comment"/>	

[Terms of Use\(http://www.freescale.com/webapp/sps/site/overview.jsp?code=TERMSOFUSE\)](http://www.freescale.com/webapp/sps/site/overview.jsp?code=TERMSOFUSE) | [Privacy\(http://www.freescale.com/webapp/sps/site/overview.jsp?code=PRIVACYPRACTICES\)](http://www.freescale.com/webapp/sps/site/overview.jsp?code=PRIVACYPRACTICES)
© 2004-2015 Freescale Semiconductor, Inc. All rights reserved.